```
RRRRRRRRRRR      MMM          MMM      SSSSSSSSSSS
RRRRRRRRRRRR     MMM          MMM      SSSSSSSSSSSS
RRRRRRRRRRRR     MMM          MMM      SSSSSSSSSSSS
RRR          RRR MMMMMMM    MMMMMMM   SSS
RRR          RRR MMMMMMM    MMMMMMM   SSS
RRR          RRR MMMMMMM    MMMMMMM   SSS
RRR          RRR MMM   MMM   MMM   MMM SSS
RRR          RRR MMM   MMM   MMM   MMM SSS
RRR          RRR MMM   MMM   MMM   MMM SSS
RRRRRRRRRRR      MMM          MMM      SSSSSSSSS
RRRRRRRRRRRR     MMM          MMM      SSSSSSSSS
RRRRRRRRRRRR     MMM          MMM      SSSSSSSSS
RRR    RRR       MMM          MMM               SSS
RRR    RRR       MMM          MMM               SSS
RRR    RRR       MMM          MMM               SSS
RRR       RRR    MMM          MMM               SSS
RRR       RRR    MMM          MMM               SSS
RRR       RRR    MMM          MMM               SSS
RRR          RRR MMM          MMM  SSSSSSSSSSS
RRR       RRR    MMM          MMM               SSS
RRR          RRR MMM          MMM  SSSSSSSSSSS
RRR          RRR MMM          MMM  SSSSSSSSSSS
RRR          RRR MMM          MMM  SSSSSSSSSSS
```

```
RRRRRRRR    MM      MM    333333    MM      MM    IIIIII      SSSSSSSS  PPPPPPPP    UU      UU  TTTTTTTTTT
RRRRRRRR    MM      MM    333333    MM      MM    IIIIII      SSSSSSSS  PPPPPPPP    UU      UU  TTTTTTTTTT
RR      RR  MMMM  MMMM  33      33  MMMM  MMMM      II      SS        PP      PP  UU      UU      TT
RR      RR  MMMM  MMMM  33      33  MMMM  MMMM      II      SS        PP      PP  UU      UU      TT
RR      RR  MM  MM  MM        33    MM  MM  MM      II      SS        PP      PP  UU      UU      TT
RR      RR  MM  MM  MM        33    MM  MM  MM      II      SS        PP      PP  UU      UU      TT
RRRRRRRR    MM      MM      33      MM      MM      II        SSSSSS  PPPPPPPP    UU      UU      TT
RRRRRRRR    MM      MM      33      MM      MM      II        SSSSSS  PPPPPPPP    UU      UU      TT
RR  RR      MM      MM        33    MM      MM      II              SS PP          UU      UU      TT
RR  RR      MM      MM        33    MM      MM      II              SS PP          UU      UU      TT
RR    RR    MM      MM  33    33    MM      MM      II              SS PP          UU      UU      TT          ....
RR    RR    MM      MM  33    33    MM      MM      II              SS PP          UU      UU      TT          ....
RR      RR  MM      MM    333333    MM      MM    IIIIII    SSSSSSSS  PP          UUUUUUUUUU      TT          ....
RR      RR  MM      MM    333333    MM      MM    IIIIII    SSSSSSSS  PP          UUUUUUUUUU      TT          ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

```
;     1          0001  0
;     2          0002  0 MODULE RM3MISPUT (LANGUAGE (BLISS32) ,
;     3          0003  0                   IDENT = 'V04-000'
;     4          0004  0                   ) =
;     5          0005  1 BEGIN
;     6          0006  1 !
;     7          0007  1 !*****************************************************************************
;     8          0008  1 !*                                                                           *
;     9          0009  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
;    10          0010  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
;    11          0011  1 !*  ALL RIGHTS RESERVED.                                                     *
;    12          0012  1 !*                                                                           *
;    13          0013  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;    14          0014  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;    15          0015  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;    16          0016  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;    17          0017  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;    18          0018  1 !*  TRANSFERRED.                                                             *
;    19          0019  1 !*                                                                           *
;    20          0020  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;    21          0021  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;    22          0022  1 !*  CORPORATION.                                                             *
;    23          0023  1 !*                                                                           *
;    24          0024  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;    25          0025  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
;    26          0026  1 !*                                                                           *
;    27          0027  1 !*                                                                           *
;    28          0028  1 !*****************************************************************************
;    29          0029  1
;    30          0030  1 !++
;    31          0031  1 !
;    32          0032  1 ! FACILITY:     RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
;    33          0033  1 !
;    34          0034  1 ! ABSTRACT:
;    35          0035  1 !                Miscellaneous put routines
;    36          0036  1 !
;    37          0037  1 !
;    38          0038  1 ! ENVIRONMENT:
;    39          0039  1 !
;    40          0040  1 !                VAX/VMS OPERATING SYSTEM
;    41          0041  1 !
;    42          0042  1 !--
;    43          0043  1
;    44          0044  1 !
;    45          0045  1 ! AUTHOR:        Christian Saether     CREATION DATE:           6-JUL-78  10:56
;    46          0046  1 !
;    47          0047  1 !
;    48          0048  1 ! MODIFIED BY:
;    49          0049  1 !
;    50          0050  1 !       V03-012 MCN0010         Maria del C. Nasr       04-Apr-1983
;    51          0051  1 !               Change linkage of RM$COMPRESS_KEY.
;    52          0052  1 !
;    53          0053  1 !       V03-011 MCN0009         Maria del C. Nasr       15-Mar-1983
;    54          0054  1 !               More linkages reorganization
;    55          0055  1 !
;    56          0056  1 !       V03-010 MCN0008         Maria del C. Nasr       22-Feb-1983
;    57          0057  1 !               RM$BLD_IDX_REC and RM$MOVE_IN_VBN are not global routines
```

```
;    58          0058  1 !               anymore, make them local.  Also, make local linkages more
;    59          0059  1 !               general, and reorganize linkages.
;    60          0060  1 !
;    61          0061  1 !       V03-009 TMK0006         Todd M. Katz            10-Sep-1982
;    62          0062  1 !               Add support for Prologue 3 SIDRs. The routines which have been
;    63          0063  1 !               modified or rewritten are: RM$RECORD_SIZE, RM$ADD_TO_ARRAY,
;    64          0064  1 !               RM$BLD_NEW_SIDR, and RM$INS_REC.
;    65          0065  1 !
;    66          0066  1 !               I also reformatted the whole module, putting the routines in
;    67          0067  1 !               alphabetical order, and redid some of the linkages.
;    68          0068  1 !
;    69          0069  1 !       V03-008 KBT0222         Keith B. Thompson       23-Aug-1982
;    70          0070  1 !               Reorganize psects
;    71          0071  1 !
;    72          0072  1 !       V03-007 TMK0005         Todd M. Katz            02-Jun-1982
;    73          0073  1 !               Implement the RMS cluster solution for next record processing.
;    74          0074  1 !               There is no longer any need to reference the NRP cell to obtain
;    75          0075  1 !               the RFA address of the current primary data record because that
;    76          0076  1 !               information is now kept locally in the IRAB as part of the next
;    77          0077  1 !               record context instead of in the NRP cell which has been
;    78          0078  1 !               deleted. Also refer to the fields IRB$_PUTUP_VBN and
;    79          0079  1 !               IRB$W_PUTUP_ID when refering to the new/changed primary data
;    80          0080  1 !               which requires the SIDR entry.
;    81          0081  1 !
;    82          0082  1 !       V03-006 KBT0066         Keith B. Thompson       17-Jun-1982
;    83          0083  1 !               Remove rm$move_sig_cnt routine
;    84          0084  1 !
;    85          0085  1 !       V03-005 TMK0004         Todd M. Katz            11-May-1982
;    86          0086  1 !               Added code for the support of two index bucket split cases
;    87          0087  1 !               which are currently not supported. At the present time when
;    88          0088  1 !               one of these split cases occur, the routines in this module are
;    89          0089  1 !               never called, but instead, a higher routine returns an
;    90          0090  1 !               alternate success status of index not updated. These two split
;    91          0091  1 !               cases are: two-pass two-bucket with empty bucket split case
;    92          0092  1 !               and two-pass multi-bucket with empty bucket split case. Support
;    93          0093  1 !               has been added for all prologue versions. The routines which
;    94          0094  1 !               have been modified are RM$RECORD_SIZE, RM$SHFT_VBNS,
;    95          0095  1 !               RM$BLD_IDX_REC, and RM$V3_IDX_REC. These routines have been
;    96          0096  1 !               modified so that they can perform their individual functions
;    97          0097  1 !               during each of the two individual passes required to correctly
;    98          0098  1 !               update the index during these two-pass empty bucket index
;    99          0099  1 !               bucket split cases. During the first pass, these routines
;   100          0100  1 !               function on the old (left) index bucket, and during the second
;   101          0101  1 !               pass these routines function on the new (right) index bucket.
;   102          0102  1 !
;   103          0103  1 !       V03-004 TMK0003         Todd M. Katz            26-Apr-1982
;   104          0104  1 !               Redid the way all empty bucket splits are done for prologue
;   105          0105  1 !               3 files. There were many errors in the way they were being done
;   106          0106  1 !               especially in those cases when a decision was made not to swing
;   107          0107  1 !               the level one index pointer from the old (left) bucket to a
;   108          0108  1 !               new (middle) bucket because the downpointer was not currently
;   109          0109  1 !               pointing to the left bucket. Changes were made to the routine
;   110          0110  1 !               RM$RECORD_SIZE so that the size of the new index record(s)
;   111          0111  1 !               could be correctly computed for all empty bucket involving
;   112          0112  1 !               index updates. Changes were made to the routine RM$SHFT_VBNS
;   113          0113  1 !               so that the spreading apart of VBNs during index updating was
;   114          0114  1 !               performed correctly for all index updates involving empty data
```

```
;  115     0115  1 !                buckets. Finally, the routine RM$V3_IDX_REC was modified to
;  116     0116  1 !                correctly perform all index updates involving empty buckets,
;  117     0117  1 !                and most especially those updates were no pointer is to be
;  118     0118  1 !                swung because the level one index down pointer does not point
;  119     0119  1 !                to the old (leftmost) data bucket. These changes will be
;  120     0120  1 !                included as a patch on the V3.1 update floopy.
;  121     0121  1 !
;  122     0122  1 !        V03-003 TMK0002         Todd M. Katz            13-Apr-1982
;  123     0123  1 !                Fixed a bug in RM$RECORD_SIZE. This bug only showed up when
;  124     0124  1 !                a multi-bucket data level split of a prolog 3 file with index
;  125     0125  1 !                compression required updating of the level 1 index with two
;  126     0126  1 !                index records. The size of these two index records was being
;  127     0127  1 !                incorrectly computed. I was refering to keybuffer 5 (containing
;  128     0128  1 !                the compressed key of the first record) in one instance when
;  129     0129  1 !                I should have been refering to keybuffer 4 (containing the
;  130     0130  1 !                compressed key of the second record), and thus the combined
;  131     0131  1 !                size of the two index records was being incorrectly computed.
;  132     0132  1 !
;  133     0133  1 !        V03-002 LJA0007         Laurie Anderson         25-Mar-1982
;  134     0134  1 !                Change KBUFSZ to reference a macro when computing buffer
;  135     0135  1 !                size and make IFB$B_KBUFSZ a word, now: IFB$W_KBUFSZ.
;  136     0136  1 !
;  137     0137  1 !        V03-001 TMK0001         Todd M. Katz            03-Mar-1982
;  138     0138  1 !                Fixed a bug in the determination of the record size of
;  139     0139  1 !                a two-bucket with empty bucket bucket split case involving
;  140     0140  1 !                prolog 3 index buckets with compressed keys.  In such a case
;  141     0141  1 !                there is no key to be inserted, but due to insufficient
;  142     0142  1 !                checks the routine RM$V3KEY_SZ was being called as if there
;  143     0143  1 !                was a key. This routine thinks there is a key in keybuffer 2
;  144     0144  1 !                and attempts to move it into keybuffer 5 so it can compress
;  145     0145  1 !                it. Since there isn't a key, it ends up moving garbage into
;  146     0146  1 !                the keybuffer and into the RLB which follows it guarenteeing an
;  147     0147  1 !                access violation the next time RM$UNLOCK is called. The fix
;  148     0148  1 !                is to make sure that this routine will never be called for
;  149     0149  1 !                this particluar prolog 3 bucket split case.
;  150     0150  1 !
;  151     0151  1 !
;  152     0152  1 !        V02-022 KPL0001         Peter Lieberwirth       2-Mar-1982
;  153     0153  1 !                Fix linkage from v02-021
;  154     0154  1 !
;  155     0155  1 !        V02-021 TMK0005         Todd M. Katz            02-Mar-1982
;  156     0156  1 !                Made several changes to the routines in this module:
;  157     0157  1 !
;  158     0158  1 !                1. RM$V3KEY_SZ is no longer a global routine.
;  159     0159  1 !
;  160     0160  1 !                2. Added a symbol to RM$FOOLED_YUH so that reprobing of the
;  161     0161  1 !                   RAB would not be neccessairy.
;  162     0162  1 !
;  163     0163  1 !                3. Added support for rear-end truncation of keys in prolog 3
;  164     0164  1 !                   buckets containing compressed keys. This included changes in
;  165     0165  1 !                   the way the size of two compressed keys (from a multibucket
;  166     0166  1 !                   split) is computed in RM$RECORD_SIZE, changes in the way the
;  167     0167  1 !                   size of a single key is determined in RM$V3KEY_SZ, and
;  168     0168  1 !                   changes in the way in which keys are inserted into prolog 3
;  169     0169  1 !                   buckets (with compressed keys) in RM$ADD_V3KEY. All these
;  170     0170  1 !                   changes reflect the fact that compressed keys are now
;  171     0171  1 !                   rear-end truncated as well as front-end compressed.
```

```
;  172     0172  1 ;
;  173     0173  1 ;       V02-020 TMK0004         Todd M. Katz            12-Feb-1982
;  174     0174  1 ;               Add checks for a multibucket split in RM$RECORD_SIZE and
;  175     0175  1 ;               RM$BLD_IDX_REC in all instances when it isn't absolutely
;  176     0176  1 ;               clear whether RMS is currently processing a two-pass two-bucket
;  177     0177  1 ;               split case or a two-pass multibucket split case (where both
;  178     0178  1 ;               records are to go in the same bucket).
;  179     0179  1 ;
;  180     0180  1 ;       V02-019 TMK0003         Todd M. Katz            08-Feb-1982
;  181     0181  1 ;               Document what is going on in RM$RECORD_SIZE so that someone
;  182     0182  1 ;               besides me might possibly understand what is going on from
;  183     0183  1 ;               the comments. The current comments are both confusing and
;  184     0184  1 ;               wrong.
;  185     0185  1 ;
;  186     0186  1 ;       V02-018 TMK0002         Todd M. Katz            30-Jan-1982
;  187     0187  1 ;               Made many changes to several of these miscellaneous routines
;  188     0188  1 ;               inorder to fix a variety of bugs, and to add functionality
;  189     0189  1 ;               that was lacking. I will list the reasons for the changes
;  190     0190  1 ;               made, and how the routines were changed.
;  191     0191  1 ;
;  192     0192  1 ;                   1. Added REC_SZ as an arguement to both RM$BLD_IDX_REC
;  193     0193  1 ;                      and RM$V3_IDX_REC. It was require in the former to pass
;  194     0194  1 ;                      it to the latter, and in the latter to fix the prolog 3
;  195     0195  1 ;                      multibucket split trailing key recompression bug described
;  196     0196  1 ;                      below.
;  197     0197  1 ;
;  198     0198  1 ;                   2. Fixed a prolog 3 multibucket split key recompression bug.
;  199     0199  1 ;                      When a multibucket split occurs, and the index must be
;  200     0200  1 ;                      updated with two keys, the key at the insertion point
;  201     0201  1 ;                      must have its front compression redetermined. However,
;  202     0202  1 ;                      the key recompression routine, RM$RECOMPR_KEY, is unable
;  203     0203  1 ;                      to recompute the front compression of a key if its
;  204     0204  1 ;                      current front compression is not equal to the front
;  205     0205  1 ;                      compression of the key preceeding it. If it has greater
;  206     0206  1 ;                      compression the index is out-of-order, and if it has less,
;  207     0207  1 ;                      it assumes there is no need for recompression. Well, if
;  208     0208  1 ;                      the two keys BB AND BC were to be inserted between AA and
;  209     0209  1 ;                      BD, the trailing key BD was not being recompressed because
;  210     0210  1 ;                      its current fron compression (0) was less than the
;  211     0211  1 ;                      compression of the record preceeding it (1) even though
;  212     0212  1 ;                      it should be recompressed to have a new front compression
;  213     0213  1 ;                      value of 1. The fix was that on a multibucket split, the
;  214     0214  1 ;                      first key is inserted and used to recompress what will
;  215     0215  1 ;                      become the trailing key, and then the second key is inserted
;  216     0216  1 ;                      and used to recompress the trailing key. This required
;  217     0217  1 ;                      changes to RM$ADD_V3KEY, so that recompression is NOT
;  218     0218  1 ;                      performed if the key being inserted is the first of two
;  219     0219  1 ;                      keys, and changes to RM$V3_IDX_REC, to recompress what
;  220     0220  1 ;                      will become the trailing key after the first of two keys
;  221     0221  1 ;                      is inserted, based on the first key that is inserted. In
;  222     0222  1 ;                      addition, I also added the change such that if the first
;  223     0223  1 ;                      of two compressed keys to be inserted is zero front
;  224     0224  1 ;                      compressed, then the pointer to the last noncompressed key
;  225     0225  1 ;                      is incremented to this key to facilitate the determination
;  226     0226  1 ;                      of the front compression of the next key when it is
;  227     0227  1 ;                      inserted.
;  228     0228  1 ;
```

```
;  229     0229  1 !
;  230     0230  1 !       3. Added code to restrict when trailing key recompression
;  231     0231  1 !          is required. If one or two keys are inserted into a
;  232     0232  1 !          prolog 3 index bucket containing compressed keys at the
;  233     0233  1 !          beginning of the bucket's freespace, there is no need to
;  234     0234  1 !          redetermine the front compression of the trailing key
;  235     0235  1 !          because there is no trailing key! Formerly, the front
;  236     0236  1 !          compression of the trailing key was blindly determined
;  237     0237  1 !          as part of the process of inserting any compressed key
;  238     0238  1 !          regardless of whether tnere was any key that trailed the
;  239     0239  1 !          key inserted or not.
;  240     0240  1 !
;  241     0241  1 !       4. Changed the decision by which the keys and VBNs in prolog
;  242     0242  1 !          3 index buckets are shifted in order to make room for the
;  243     0243  1 !          insertion of a new key-VBN pair. At the present time, if
;  244     0244  1 !          the key/VBN is to be added as the hish order key in the
;  245     0245  1 !          bucket, neither the keys nor the VBNs are shifted since
;  246     0246  1 !          both "grow" inward towards the center. An additional check
;  247     0247  1 !          is added to also not shift the keys or VBNs, if the
;  248     0248  1 !          number of bytes by which they are to be shifted "inwards"
;  249     0249  1 !          is 0. This will occur when the split point of an index
;  250     0250  1 !          bucket matches the insertion point, and just the VBN of
;  251     0251  1 !          the new lower order key of the new (right) bucket is to
;  252     0252  1 !          be updated to its new value.
;  253     0253  1 !
;  254     0254  1 !       5. Added code necesaary for support of a Prolog 1 & 2 split
;  255     0255  1 !          case. When the insertion point of a multibucket split is
;  256     0256  1 !          found to be at the split point, but there is sufficient room
;  257     0257  1 !          in an index bucket for only two index records, and the
;  258     0258  1 !          insertion point happens to come at the bucket's beginning,
;  259     0259  1 !          both of the old index records must be moved out of the old
;  260     0260  1 !          bucket to make room for the two new index records which are
;  261     0261  1 !          both put in the old bucket. This index bucket spiit case
;  262     0262  1 !          required code to be added to RM$RECORD_SIZE and
;  263     0263  1 !          RM$BLD_IDX_REC. As this split case is handled by making two
;  264     0264  1 !          passes, one for insertion of the two new index records in
;  265     0265  1 !          the old index bucket, and the second to update a VBN pointer
;  266     0266  1 !          in the low order key of the new bucket, both of these
;  267     0267  1 !          routines had to be modified, to take the proper action for
;  268     0268  1 !          each of the passes of this case. RM$BLD_IDX_REC oversees
;  269     0269  1 !          addition of the both keys and a VBN during the first pass,
;  270     0270  1 !          and updates a VBN during the second, while RM$RECORD_SIZE
;  271     0271  1 !          makes sure to return the proper size of the index record
;  272     0272  1 !          to be added during each pass.
;  273     0273  1 !
;  274     0274  1 !       6. There are three bucket split cases for when the insertion
;  275     0275  1 !          point and the bucket split point are the same and no empty
;  276     0276  1 !          buckets are involved. For all three cases, two passes must
;  277     0277  1 !          be made, and for all three cases no code existed for Prolog
;  278     0278  1 !          3 buckets. Changes were made to RM$RECORD_SIZE, RM$SHFT_VBNS
;  279     0279  1 !          and RM$V3_IDX_REC to support these three bucket split cases.
;  280     0280  1 !          The changes made to RM$V3_IDX_REC allow the routine to
;  281     0281  1 !          recognize these three two-pass bucket split cases, which of
;  282     0282  1 !          the two passes is in progress, and to take the appropriate
;  283     0283  1 !          action. As the VBNs in prolog 3 buckets are order from
;  284     0284  1 !          right to left and "grow inwards", if a index record with a
;  285     0285  1 !          key value less than the current high key value of the bucket
;                            is inserted, the VBNs must be shifted to make room for the
```

```
;  286     0286  1 !                VBN associated with the just inserted key. The routine which
;  287     0287  1 !                performs this shifting (RM$SHFT_VBNS) had to be modified to
;  288     0288  1 !                recognize these three bucket split cases, which of the two
;  289     0289  1 !                passes was current, and perform any required VBN shifting.
;  290     0290  1 !                Finally, RM$RECORD_SIZE had to be similary modified so that
;  291     0291  1 !                it would return the size of the index record to be inserted
;  292     0292  1 !                into the "current" bucket. The size, of course, as well as
;  293     0293  1 !                the current bucket depends upon the split case, and which
;  294     0294  1 !                of the two passes is currently under way. The three bucket
;  295     0295  1 !                split cases and their effect on these routines are as
;  296     0296  1 !                follows:
;  297     0297  1 !
;  298     0298  1 !                TWO-BUCKET SPLIT:
;  299     0299  1 !                During the first pass (when the index record contents of the
;  300     0300  1 !                old bucket are updated) just a key is added. The size of the
;  301     0301  1 !                record is the size of the key, and no VBNs need be shifted.
;  302     0302  1 !                During the second pass (when the index record contents of
;  303     0303  1 !                the new index bucket are appropriately modified), just a VBN
;  304     0304  1 !                is updated. The size of the record is the difference between
;  305     0305  1 !                the VBN size required to hold the old data level VBN and the
;  306     0306  1 !                size required to hold the new data level VBN. No VBN
;  307     0307  1 !                shifting is required unless the size is greater than 0.
;  308     0308  1 !
;  309     0309  1 !                THREE-BUCKET SPLIT - ONE KEY IN EACH INDEX BUCKET:
;  310     0310  1 !                During the first pass, just a key is added to the contents
;  311     0311  1 !                of the old index bucket so the record size is the size of
;  312     0312  1 !                the key and the no VBNs need be shifted. During the second
;  313     0313  1 !                pass, a new key and VBN is added as the low order key in the
;  314     0314  1 !                new index bucket, and a VBN is also updated. In this case
;  315     0315  1 !                VBNs must be shifted to make room for the new VBN and any
;  316     0316  1 !                change in VBN size, and the record size includes the key
;  317     0317  1 !                added, the VBN added, and any difference in data level
;  318     0318  1 !                VBN size.
;  319     0319  1 !
;  320     0320  1 !                THREE-BUCKET SPLIT - BOTH KEYS IN OLD INDEX BUCKET:
;  321     0321  1 !                This split case only occurs when the insertion point and
;  322     0322  1 !                the split point are at the beginning of the old index
;  323     0323  1 !                bucket and both keys in the old bucket must be moved out
;  324     0324  1 !                to make room for the two new keys. During the first pass,
;  325     0325  1 !                both keys are added as is a VBN. The record size is the
;  326     0326  1 !                size of the two keys and the VBN added, and no VBN shifting
;  327     0327  1 !                need take place. During the second pass just a VBN is
;  328     0328  1 !                updated. The record size is computed to be any difference
;  329     0329  1 !                in sizes required to hold data level VBNs, and VBN shifting
;  330     0330  1 !                takes place only if there is a difference.
;  331     0331  1 !
;  332     0332  1 !        V02-017 TMK0001         Todd M. Katz            11-Jan-1982
;  333     0333  1 !                Make a change to RM$SHFT_VBNS which affects three-way bucket
;  334     0334  1 !                splits where one of the resulting buckets is empty. The
;  335     0335  1 !                routine was adjusting the offset pointer to where the
;  336     0336  1 !                VBN shift should start. This is wrong, and in fact, caused
;  337     0337  1 !                index corruption so I removed it.
;  338     0338  1 !
;  339     0339  1 !        V02-016 PSK0007         Paulina S. Knibbe       14-Dec-1981
;  340     0340  1 !                Change record_size to take BKT_ADDR as an input
;  341     0341  1 !
;  342     0342  1 !        V02-015 PSK0006         Paulina S. Knibbe       26-Oct-1981
```

```
;  343     0343  1 !               Add support for compressed indexes to rm$v3key_sz
;  344     0344  1 !
;  345     0345  1 !       V02-014 PSK0005         Paulina S. Knibbe       15-Aug-1981
;  346     0346  1 !               Fix some problems w/large VBNs in index buckets.
;  347     0347  1 !
;  348     0348  1 !       V02-013 PSK0004         Paulina S. Knibbe       29-Jul-1981
;  349     0349  1 !               Remove support for growing prologue three compressed
;  350     0350  1 !               indexes. Add code to handle indexes with no  chars
;  351     0351  1 !               compressed off of the rear.
;  352     0352  1 !
;  353     0353  1 !       V02-012 PSK0003         Paulina S. Knibbe       24-Jul-1981
;  354     0354  1 !               Fix off-by-one error in SHIFT_VBNS; other problems
;  355     0355  1 !               with growing VBNs
;  356     0356  1 !
;  357     0357  1 !       V02-011 PSK0002         Paulina S. Knibbe       13-Jul-1981
;  358     0358  1 !               Add 'RM$SHFT_VBNS' to handle spreading the VBN chain
;  359     0359  1 !               in prologue three index buckets.
;  360     0360  1 !
;  361     0361  1 !       V02-010 MCN0007         Maria del C. Nasr       28-Jun-1981
;  362     0362  1 !               Add RM$MOVE_SIG_CNT.
;  363     0363  1 !
;  364     0364  1 !       V02-009 PSK0001         Paulina S. Knibbe       08-Jun-1981
;  365     0365  1 !               Add support for prologue three index buckets:
;  366     0366  1 !               RM$RECORD_SIZE, RM$NEW_VBN_BYTES, RM$V3KEY_SZ
;  367     0367  1 !               RM$V3_IDX_REC,RM$ADD_V3VBN,RM$ADD_V3KEY
;  368     0368  1 !
;  369     0369  1 !       V02-008 MCN0006         Maria del C. Nasr       16-Mar-1981
;  370     0370  1 !               Increase size of record identifier to a word in the NRP.
;  371     0371  1 !               Change linkage to RM$RECORD_SIZE to include IFAB.
;  372     0372  1 !
;  373     0373  1 !       V02-007 REFORMAT        C D Saether     01-Aug-1980     18:10
;  374     0374  1 !
;  375     0375  1 !       V0006   CDS0077         C D SAETHER     24-JAN-1980     13:50
;  376     0376  1 !               Don't want to create dupes count field on continuation
;  377     0377  1 !               SIDR arrays either.
;  378     0378  1 !
;  379     0379  1 !       V0005   CDS0071         C D SAETHER     15-JAN-1980     12:28
;  380     0380  1 !               Fix bug creating dupes count field in SIDR records when
;  381     0381  1 !               dupes aren't allowed (rm$record_size and rm$bld_new_sidr).
;  382     0382  1 !
;  383     0383  1 ! Revision history:
;  384     0384  1 !
;  385     0385  1 !   Wendy Koenig,       24-OCT-78  14:02
;  386     0386  1 !   X0002 - MAKE CHANGES CAUSED BY SHARING CONVENTIONS
;  387     0387  1 !
;  388     0388  1 !   Christian Saether,  26-JAN-79  9:01
;  389     0389  1 !   X0003 - check down pointer before attempting to swing from empty bucket
;  390     0390  1 !
;  391     0391  1 !   Christian Saether,  1-july-79  11:00
;  392     0392  1 !   X0004 - fix nxtrecid logic in RM$INS_REC (caused re-use of ID's)
;  393     0393  1 !
;  394     0394  1 !*****
;  395     0395  1
;  396     0396  1 LIBRARY 'RMSLIB:RMS';
;  397     0397  1
;  398     0398  1 REQUIRE 'RMSSRC:RMSIDXDEF';
;  399     0463  1
```

```
;  400          0464  1 !
;  401          0465  1 ! define default psects for code
;  402          0466  1 !
;  403          0467  1
;  404          0468  1 PSECT
;  405          0469  1     CODE = RM$RMS3(PSECT_ATTR),
;  406          0470  1     PLIT = RM$RMS3(PSECT_ATTR);
;  407          0471  1
;  408          0472  1 !
;  409          0473  1 ! Linkages
;  410          0474  1
;  411          0475  1 LINKAGE
;  412          0476  1     L_COMPARE_KEY,
;  413          0477  1     L_JSB,
;  414          0478  1     L_JSB01,
;  415          0479  1     L_PRESERVE1,
;  416          0480  1     L_RABREG_567,
;  417          0481  1     L_RABREG_67,
;  418          0482  1     L_RABREG_7,
;  419          0483  1     L_REC_OVHD,
;  420          0484  1
;  421          0485  1     ! Local Linkages
;  422          0486  1     !
;  423          0487  1     RL$ADD_TO_ARRAY = JSB () :
;  424          0488  1                       GLOBAL (R_IDX_DFN, R_IRAB, R_IFAB, R_REC_ADDR),
;  425          0489  1     RL$COMMON_LINK  = JSB (STANDARD) :
;  426          0490  1                       GLOBAL (R_BKT_ADDR, R_IFAB, R_IRAB, R_IDX_DFN, R_REC_ADDR),
;  427          0491  1     RL$MOVE_IN_VBN  = JSB (REGISTER = 2) :
;  428          0492  1                       GLOBAL (R_REC_ADDR)
;  429          0493  1                       NOPRESERVE (2),
;  430          0494  1     RL$NEW_VBN_BYTES = JSB () :
;  431          0495  1                        GLOBAL (R_BKT_ADDR, R_IDX_DFN);
;  432          0496  1
;  433          0497  1 ! Forward Routines
;  434          0498  1 !
;  435          0499  1 FORWARD ROUTINE
;  436          0500  1     RM$MOVE_IN_VBN      : RL$MOVE_IN_VBN NOVALUE,
;  437          0501  1     RM$SHFT_VBNS        : RL$COMMON_LINK NOVALUE,
;  438          0502  1     RM$V3_IDX_REC       : RL$COMMON_LINK NOVALUE,
;  439          0503  1     RM$V3KEY_SZ         : RL$COMMON_LINK,
;  440          0504  1     RM$VBN_SIZE         : RL$PRESERVE1;
;  441          0505  1
;  442          0506  1 ! External Routines
;  443          0507  1 !
;  444          0508  1 EXTERNAL ROUTINE
;  445          0509  1     RM$CNTRL_ADDR       : RL$RABREG_567,
;  446          0510  1     RM$COMPARE_KEY      : RL$COMPARE_KEY,
;  447          0511  1     RM$COMPRESS_KEY     : RL$JSB01,
;  448          0512  1     RM$GETNEXT_REC      : RL$RABREG_67,
;  449          0513  1     RM$MOVE             : RL$PRESERVE1,
;  450          0514  1     RM$NOREAD_LONG      : RL$JSB,
;  451          0515  1     RM$RECOMPR_KEY      : RL$JSB01,
;  452          0516  1     RM$RECORD_VBN       : RL$PRESERVE1,
;  453          0517  1     RM$REC_OVHD         : RL$REC_OVHD,
;  454          0518  1     RM$V3_VBN           : RL$RABREG_567;
;  455          0519  1
;  456          0520  1 MACRO
```

```
;  457          0521  1      KEY_LEN      = 0,0,8,0 %,
;  458          0522  1      CMPR_CNT     = 1,0,8,0 %;
;  459          0523  1
```

```
;  461     0524  1 %SBTTL  'RM$ADD_TO_ARRAY'
;  462     0525  1 ROUTINE RM$ADD_TO_ARRAY : RL$ADD_TO_ARRAY NOVALUE =
;  463     0526  1
;  464     0527  1 !++
;  465     0528  1 !
;  466     0529  1 ! FUNCTIONAL DESCRIPTION:
;  467     0530  1 !
;  468     0531  1 !       This routine adds a new array to an existing SIDR array, and updates
;  469     0532  1 !       the size field of the SIDR.
;  470     0533  1 !
;  471     0534  1 ! CALLING SEQUENCE:
;  472     0535  1 !
;  473     0536  1 !       RM$ADD_TO_ARRAY()
;  474     0537  1 !
;  475     0538  1 ! INPUT PARAMETERS:
;  476     0539  1 !       NONE
;  477     0540  1 !
;  478     0541  1 ! IMPLICIT INPUTS:
;  479     0542  1 !
;  480     0543  1 !       IFAB                    - address of IFAB
;  481     0544  1 !           IFB$B_PLG_VER       - prologue version of the file
;  482     0545  1 !
;  483     0546  1 !       IRAB                    - address of IRAB
;  484     0547  1 !           IRB$L_LST_REC       - address of beginning of SIDR
;  485     0548  1 !           IRB$W_PUTUPD_ID     - ID of UDR to put into record pointer
;  486     0549  1 !           IRB$L_PUTUPD_VBN    - VBN of UDR to put into record pointer
;  487     0550  1 !
;  488     0551  1 !       REC_ADDR                - address of where new array element is to go
;  489     0552  1 !
;  490     0553  1 ! OUTPUT PARAMETERS:
;  491     0554  1 !       NONE
;  492     0555  1 !
;  493     0556  1 ! IMPLICIT OUTPUTS:
;  494     0557  1 !
;  495     0558  1 !       REC_ADDR points         - address of SIDR's size field
;  496     0559  1 !
;  497     0560  1 ! ROUTINE VALUE:
;  498     0561  1 !       NONE
;  499     0562  1 !
;  500     0563  1 ! SIDE EFFECTS:
;  501     0564  1 !       NONE
;  502     0565  1 !
;  503     0566  1 !--
;  504     0567  1
;  505     0568  2     BEGIN
;  506     0569  2
;  507     0570  2     EXTERNAL REGISTER
;  508     0571  2         R_IDX_DFN,
;  509     0572  2         R_IFAB_STR,
;  510     0573  2         R_IRAB_STR,
;  511     0574  2         R_REC_ADDR_STR;
;  512     0575  2
;  513     0576  2     GLOBAL REGISTER
;  514     0577  2         R_RAB,
;  515     0578  2         R_IMPURE,
;  516     0579  2         R_BDB;
;  517     0580  2
```

```
;  518          0581  2     LOCAL
;  519          0582  2         VBN_SIZE        : BYTE,
;  520          0583  2         TOTAL_SIZE      : BYTE;
;  521          0584  2
;  522          0585  2     ! Determine what will be the size of the record pointer's VBN.
;  523          0586  2     !
;  524          0587  2     VBN_SIZE = RM$VBN_SIZE (.IRAB[IRB$L_PUTUP_VBN]);
;  525          0588  2
;  526          0589  2     ! Set up the pointer size field in the new array element's control byte.
;  527          0590  2     !
;  528          0591  2     (.REC_ADDR)<0, 8> = .VBN_SIZE - 2;
;  529          0592  2     REC_ADDR = .REC_ADDR + 1;
;  530          0593  2
;  531          0594  2     ! Insert the record pointer's ID into the new SIDR array element, and
;  532          0595  2     ! compute the overall size of the new SIDR array element. Both of these
;  533          0596  2     ! steps are prologue dependent.
;  534          0597  2     !
;  535          0598  2     IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
;  536          0599  2     THEN
;  537          0600  3         BEGIN
;  538          0601  3         (.REC_ADDR)<0, 8> = .IRAB[IRB$W_PUTUP_ID];
;  539          0602  3         REC_ADDR = .REC_ADDR + 1;
;  540          0603  3         TOTAL_SIZE = .VBN_SIZE + IRC$C_DATOVHDSZ;
;  541          0604  3         END
;  542          0605  2     ELSE
;  543          0606  3         BEGIN
;  544          0607  3         (.REC_ADDR)<0, 16> = .IRAB[IRB$W_PUTUP_ID];
;  545          0608  3         REC_ADDR = .REC_ADDR + 2;
;  546          0609  3         TOTAL_SIZE = .VBN_SIZE + IRC$C_DATOVHSZ3;
;  547          0610  2         END;
;  548          0611  2
;  549          0612  2     ! Insert the record pointer's VBN into the new SIDR array element.
;  550          0613  2     !
;  551          0614  2     (.REC_ADDR)<0, .VBN_SIZE*8> = .IRAB[IRB$L_PUTUP_VBN];
;  552          0615  2
;  553          0616  2     ! Position to the size field of the SIDR, and update it to reflect
;  554          0617  2     ! the addition of the new array element.
;  555          0618  2     !
;  556          0619  2     REC_ADDR = .IRAB[IRB$L_LST_REC];
;  557          0620  2     REC_ADDR = .REC_ADDR + RM$REC_OVHD(-1) - IRC$C_DATSZFLD;
;  558          0621  2     (.REC_ADDR)<0, 16> = .(.REC_ADDR)<0, 16> + .TOTAL_SIZE;
;  559          0622  1     END;


                                                    .TITLE  RM3MISPUT
                                                    .IDENT  \V04-000\

                                                    .EXTRN  RM$CNTRL_ADDR, RM$COMPARE_KEY
                                                    .EXTRN  RM$COMPRESS_KEY
                                                    .EXTRN  RM$GETNEXT_REC, RM$MOVE
                                                    .EXTRN  RM$NOREAD_LONG, RM$RECOMPR_KEY
                                                    .EXTRN  RM$RECORD_VBN, RM$REC_OVHD
                                                    .EXTRN  RM$V3_VBN

                                                    .PSECT  RM$RMS3,NOWRT,  GBL,  PIC,2

                                0910    8F  BB 00000 RM$ADD_TO_ARRAY:
```

```
                                                                 PUSHR   #^M<R4,R8,R11>                          ; 0525
                                                                 PUSHL   120(IRAB)                              ; 0587
                                                                 BSBW    RM$VBN_SIZE                            ;
                                                                 ADDL2   #4, SP                                 ;
                                                                 MOVB    R0, VBN_SIZE                           ;
                                                                 SUBB3   #2, VBN_SIZE, (REC_ADDR)+              ; 0591
                                                                 CMPB    183(IFAB), #3                          ; 0598
                                                                 BGEQU   1$                                     ;
                                                                 MOVB    128(IRAB), (REC_ADDR)+                 ; 0601
                                                                 ADDB3   #2, VBN_SIZE, TOTAL_SIZE               ; 0603
                                                                 BRB     2$                                     ; 0598
                                                                 MOVW    128(IRAB), (REC_ADDR)+                 ; 0607
                                                                 ADDB3   #3, VBN_SIZE, TOTAL_SIZE               ; 0609
                                                                 MOVZBL  VBN_SIZE, R0                           ; 0614
                                                                 MULL2   #8, R0                                 ;
                                                                 INSV    120(IRAB), 40, R0, (REC_ADDR)          ;
                                                                 MOVL    76(IRAB), REC_ADDR                     ; 0619
                                                                 MNEGL   #1, R1                                 ; 0620
                                                                 BSBW    RM$REC_OVHD                            ;
                                                                 MOVAB   -2(R0)[REC_ADDR], REC_ADDR             ;
                                                                 MOVZBL  TOTAL_SIZE, R0                         ; 0621
                                                                 ADDW2   R0, (REC_ADDR)                         ;
                                                                 POPR    #^M<R4,R8,R11>                         ; 0622
                                                                 RSB                                            ;

; Routine Size:  85 bytes,    Routine Base:  RM$RMS3 + 0000
```

```
;  561          0623  1 %SBTTL  'RM$ADD_V3KEY'
;  562          0624  1 ROUTINE RM$ADD_V3KEY (KEY_ADDR) : RL$COMMON_LINK NOVALUE =
;  563          0625  1 !++
;  564          0626  1 !
;  565          0627  1 ! RM$ADD_V3KEY
;  566          0628  1 !
;  567          0629  1 !       This routine inserts the key at KEY_ADDR into the bucket at REC_ADDR
;  568          0630  1 !       The contents of the bucket have already been spread apart. The
;  569          0631  1 !       trailing record is recompressed if necessary.
;  570          0632  1 !
;  571          0633  1 ! CALLING SEQUENCE:
;  572          0634  1 !
;  573          0635  1 !       RM$ADD_V3KEY()
;  574          0636  1 !
;  575          0637  1 ! IMPLICIT INPUTS:
;  576          0638  1 !       IFAB
;  577          0639  1 !       REC_ADDR
;  578          0640  1 !       BKT_ADDR
;  579          0641  1 !       IDX_DFN
;  580          0642  1 !       IRAB[IRB$V_BIG_SPLIT]
;  581          0643  1 !       IRAB[IRB$V_EMPTY_BKT]
;  582          0644  1 !
;  583          0645  1 ! OUTPUT PARAMETERS:
;  584          0646  1 !
;  585          0647  1 !       NONE
;  586          0648  1 !
;  587          0649  1 ! IMPLICIT OUTPUTS:
;  588          0650  1 !
;  589          0651  1 !       REC_ADDR points past the inserted key
;  590          0652  1 !       IRAB[IRB$L_LST_NCMP] may possibly be updated
;  591          0653  1 !
;  592          0654  1 ! ROUTINE VALUE:
;  593          0655  1 !
;  594          0656  1 !       NONE
;  595          0657  1 !
;  596          0658  1 !--
;  597          0659  1
;  598          0660  2     BEGIN
;  599          0661  2
;  600          0662  2     MAP
;  601          0663  2         KEY_ADDR : REF BBLOCK;
;  602          0664  2
;  603          0665  2     EXTERNAL REGISTER
;  604          0666  2         R_IDX_DFN_STR,
;  605          0667  2         R_IFAB_STR,
;  606          0668  2         R_REC_ADDR_STR,
;  607          0669  2         R_BKT_ADDR_STR,
;  608          0670  2         R_IRAB_STR;
;  609          0671  2
;  610          0672  2     GLOBAL REGISTER
;  611          0673  2         R_BDB,
;  612          0674  2         R_IMPURE,
;  613          0675  2         R_RAB;
;  614          0676  2
;  615          0677  2     IF NOT .IDX_DFN [IDX$V_IDX_COMPR]
;  616          0678  2     THEN
;  617          0679  2
```

```
;  618          0680  2          ! Fixed length index key.
;  619          0681  2          !
;  620          0682  3          BEGIN
;  621          0683  3          REC_ADDR = RM$MOVE (.IDX_DFN [IDX$B_KEYSZ], .KEY_ADDR, .REC_ADDR);
;  622          0684  3          RETURN
;  623          0685  2          END;
;  624          0686  2
;  625          0687  2      ! Compressed index key
;  626          0688  2      ! First fill in the length and compression count
;  627          0689  2      !
;  628          0690  2      RM$COMPRESS_KEY (.KEY_ADDR);
;  629          0691  2
;  630          0692  2      ! Now move in the new key including the two bytes of compression overhead.
;  631          0693  2      !
;  632          0694  2      RM$MOVE (.KEY_ADDR[KEY_LEN] + 2,
;  633          0695  2               .KEY_ADDR,
;  634          0696  2               .REC_ADDR);
;  635          0697  2
;  636          0698  2      ! Fix up next record (front compression might change). This will be done
;  637          0699  2      ! only when a single a key is to be added, or when the second of two keys
;  638          0700  2      ! is being added. A key must also trail the key just inserted for this
;  639          0701  2      ! recompression to be performed.
;  640          0702  2      !
;  641          0703  3      IF (NOT .IRAB[IRB$V_BIG_SPLIT] OR  .IRAB[IRB$V_EMPTY_BKT])
;  642          0704  2          AND
;  643          0705  3          (.REC_ADDR + .REC_ADDR[KEY_LEN] + 2
;  644          0706  3                  LSSU .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
;  645          0707  2      THEN
;  646          0708  2          RM$RECOMPR_KEY (.REC_ADDR, .REC_ADDR + .REC_ADDR [KEY_LEN] + 2);
;  647          0709  2
;  648          0710  2      ! If the key to be added is the first of two keys (from a multibucket split)
;  649          0711  2      ! and the front compression of the key just added is 0, increment the pointer
;  650          0712  2      ! to the last noncompressed key to this key to make the determination of the
;  651          0713  2      ! front compression of the second of the two keys easier.
;  652          0714  2      !
;  653          0715  3      IF .IRAB[IRB$V_BIG_SPLIT] AND (.REC_ADDR[CMPR_CNT] EQLU 0)
;  654          0716  2      THEN
;  655          0717  2          IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  656          0718  2
;  657          0719  2      ! Reset REC_ADDR to point to the key which follows the key(s) just inserted
;  658          0720  2      !
;  659          0721  2      REC_ADDR = .REC_ADDR + .REC_ADDR [KEY_LEN] + 2;
;  660          0722  2
;  661          0723  2      RETURN
;  662          0724  2
;  663          0725  1      END;


                         0910   8F  BB 00000 RM$ADD_V3KEY:
                                              PUSHR   #^M<R4,R8,R11>                  ; 0624
14       1C   A7                03  E0 00004          BBS     #3, 28(IDX_DFN), 1$     ; 0677
                                56  DD 00009          PUSHL   REC_ADDR                ; 0683
                                14  AE  DD 0000B      PUSHL   KEY_ADDR                ;
                          7E    20  A7  9A 0000E      MOVZBL  32(IDX_DFN), -(SP)      ;
```

```
                                0000G 30 00012          BSBW    RM$MOVE                                         ;
                             5E   0C  C0 00015          ADDL2   #12, SP                                         ;
                             56   50  D0 00018          MOVL    R0, REC_ADDR                                    ;
                                  5C  11 0001B          BRB     5$                                              ; 0682
                             50   10  AE D0 0001D 1$:    MOVL    KEY_ADDR, R0                                    ; 0690
                                0000G 30 00021          BSBW    RM$COMPRESS_KEY
                                  56  DD 00024          PUSHL   REC_ADDR                                        ; 0696
                                14    AE DD 00026       PUSHL   KEY_ADDR                                        ; 0695
                             7E 18    BE 9A 00029       MOVZBL  @KEY_ADDR, -(SP)                                ; 0694
                             6E   02  C0 0002D          ADDL2   #2, (SP)
                                0000G 30 00030          BSBW    RM$MOVE
                             5E   0C  C0 00033          ADDL2   #12, SP                                         ;
                 05      44  A9   02  E1 00036          BBC     #2, 68(IRAB), 2$                                ; 0703
                 22      44  A9   06  E1 0003B          BBC     #6, 68(IRAB), 3$
                             50   66  9A 00040 2$:      MOVZBL  (REC_ADDR), R0                                  ; 0705
                             51 02 A046 9E 00043        MOVAB   2(R0)[REC_ADDR], R1
                             50   04  A5  3C 00048      MOVZWL  4(BKT_ADDR), R0                                 ; 0706
                             50   55  C0 0004C          ADDL2   BKT_ADDR, R0
                             50   51  D1 0004F          CMPL    R1, R0
                                  0E  1E 00052          BGEQU   3$
                             50   66  9A 00054          MOVZBL  (REC_ADDR), R0                                  ; 0708
                             51 02 A046 9E 00057        MOVAB   2(R0)[REC_ADDR], R1
                             50   56  D0 0005C          MOVL    REC_ADDR, R0
                                0000G 30 0005F          BSBW    RM$RECOMPR_KEY
                 0A      44  A9   02  E1 00062 3$:      BBC     #2, 68(IRAB), 4$                                ; 0715
                           01    A6  95 00067          TSTB    1(REC_ADDR)
                                 05  12 0006A          BNEQ    4$
                       0098    C9   56  D0 0006C       MOVL    REC_ADDR, 152(IRAB)                             ; 0717
                             50   66  9A 00071 4$:      MOVZBL  (REC_ADDR), R0                                  ; 0721
                             56 02 A046 9E 00074        MOVAB   2(R0)[REC_ADDR], REC_ADDR
                           0910   8F  BA 00079 5$:      POPR    #^M<R4,R8,R11>                                  ; 0725
                                      05 0007D          RSB                                                     ;

; Routine Size:  126 bytes,    Routine Base:  RM$RMS3 + 0055
```

```
;  665     0726  1 %SBTTL  'RM$ADD_V3VBN'
;  666     0727  1 ROUTINE RM$ADD_V3VBN (VBN) : RL$COMMON_LINK NOVALUE =
;  667     0728  1 !++
;  668     0729  1 !
;  669     0730  1 ! RM$ADD_V3VBN                                                                    ; Rc
;  670     0731  1 !
;  671     0732  1 !       This routine adds a VBN to the chain at the end of the bucket.
;  672     0733  1 !       If the VBN is longer than the current VBN size all VBNs grow
;  673     0734  1 !       by the appropriate number of bytes.
;  674     0735  1 !
;  675     0736  1 ! CALLING SEQUENCE:
;  676     0737  1 !
;  677     0738  1 !       RM$ADD_V3VBN(.VBN)
;  678     0739  1 !
;  679     0740  1 ! IMPLICIT INPUTS:
;  680     0741  1 !       IRAB
;  681     0742  1 !       BKT_ADDR
;  682     0743  1 !       IDX_DFN
;  683     0744  1 !
;  684     0745  1 ! OUTPUT PARAMETERS:
;  685     0746  1 !
;  686     0747  1 !       NONE
;  687     0748  1 !
;  688     0749  1 ! IMPLICIT OUTPUTS:
;  689     0750  1 !
;  690     0751  1 !
;  691     0752  1 ! ROUTINE VALUE:
;  692     0753  1 !
;  693     0754  1 !       NONE
;  694     0755  1 !
;  695     0756  1 !--
;  696     0757  1
;  697     0758  2     BEGIN
;  698     0759  2
;  699     0760  2     MACRO
;  700     0761  2         OFFSET = 0,0,16,0 %;
;  701     0762  2
;  702     0763  2     EXTERNAL REGISTER
;  703     0764  2         R_IDX_DFN_STR,
;  704     0765  2         R_IRAB_STR,
;  705     0766  2         R_REC_ADDR_STR,
;  706     0767  2         R_IFAB_STR,
;  707     0768  2         R_BKT_ADDR_STR;
;  708     0769  2
;  709     0770  2     GLOBAL REGISTER
;  710     0771  2         R_BDB,
;  711     0772  2         R_RAB,
;  712     0773  2         R_IMPURE;
;  713     0774  2
;  714     0775  2     LOCAL
;  715     0776  2         NEW_SIZE,
;  716     0777  2         OLD_SIZE;
;  717     0778  2
;  718     0779  2      EW_SIZE = RM$VBN_SIZE (.VBN);
;  719     0780  2     OLD_SIZE = .BKT_ADDR [BKT$V_PTR_SZ] + 2;
;  720     0781  2
;  721     0782  2     IF .NEW_SIZE GTRU .OLD_SIZE
```

```
;  722     0783  2     THEN
;  723     0784  2
;  724     0785  2         ! We need to shuffle all of the VBNs down
;  725     0786  2         ! First figure out where to move them.
;  726     0787  2         !
;  727     0788  3         BEGIN
;  728     0789  3
;  729     0790  3         LOCAL
;  730     0791  3             OLD_END     : REF BBLOCK,
;  731     0792  3             NUM_RECS,
;  732     0793  3             NEW_END,
;  733     0794  3             SAVE;
;  734     0795  3
;  735     0796  3         OLD_END = .BKT_ADDR  + (.IDX_DFN [IDX$B_IDXBKTSZ] * 512) - BKT$C_ENDOVHD;
;  736     0797  3         NUM_RECS = (.OLD_END - (.BKT_ADDR+.OLD_END [OFFSET])) / .OLD_SIZE;
;  737     0798  3         NEW_END = .OLD_END - (.NUM_RECS * .NEW_SIZE);
;  738     0799  3
;  739     0800  3         IF .BKT_ADDR [BKT$B_LEVEL] EQLU 0
;  740     0801  3             AND .IDX_DFN [IDX$V_DUPKEYS]
;  741     0802  3         THEN
;  742     0803  3             NEW_END = .NEW_END - 4;
;  743     0804  3
;  744     0805  3         OLD_END [OFFSET] = .NEW_END - .BKT_ADDR - 1;
;  745     0806  3
;  746     0807  3         ! Now actually move them
;  747     0808  3         !
;  748     0809  3         SAVE = .IRAB [IRB$L_REC_COUNT];
;  749     0810  3         IRAB [IRB$L_REC_COUNT] = .NUM_RECS - 1;
;  750     0811  3
;  751     0812  3         WHILE .IRAB [IRB$L_REC_COUNT] GEQ 0
;  752     0813  3         DO
;  753     0814  4             BEGIN
;  754     0815  4             (.NEW_END) <0,8 * .NEW_SIZE> = .(RM$CNTRL_ADDR())<0,8*.OLD_SIZE>;
;  755     0816  4             IRAB [IRB$L_REC_COUNT] = .IRAB [IRB$L_REC_COUNT] - 1;
;  756     0817  4             NEW_END = .NEW_END + .NEW_SIZE
;  757     0818  3             END;
;  758     0819  3
;  759     0820  3         IRAB [IRB$L_REC_COUNT] = .SAVE;
;  760     0821  3
;  761     0822  3         ! Update the VBN size
;  762     0823  3         !
;  763     0824  3         BKT_ADDR [BKT$V_PTR_SZ] = .NEW_SIZE - 2
;  764     0825  3
;  765     0826  2         END;
;  766     0827  2
;  767     0828  2     ! Actually add the new VBN
;  768     0829  2     !
;  769     0830  3     BEGIN
;  770     0831  3
;  771     0832  3     LOCAL
;  772     0833  3         ADDR;
;  773     0834  3
;  774     0835  3     ADDR = RM$CNTRL_ADDR();
;  775     0836  3     (.ADDR) <0,.NEW_SIZE * 8> = .VBN
;  776     0837  3
;  777     0838  2     END; ! Of local ADDR
;  778     0839  2
```

```
;  779          0840  1      END;

                                091C   8F  BB 00000 RM$ADD_V3VBN:
                                                    PUSHR   #^M<R2,R3,R4,R8,R11>                    ; 0727
                              5E              0C  C2 00004          SUBL2   #12, SP
                                24            AE  DD 00007          PUSHL   VBN                     ; 0779
                                          0000V 30 0000A          BSBW    RM$VBN_SIZE
                              5E              04  C0 0000D          ADDL2   #4, SP
                              6E              50  D0 00010          MOVL    R0, NEW_SIZE
                      52      0D      A5      02  EF 00013          EXTZV   #3, #2, 13(BKT_ADDR), OLD_SIZE  ; 0780
                              52              02  C0 00019          ADDL2   #2, OLD_SIZE
                              52              6E  D1 0001C          CMPL    NEW_SIZE, OLD_SIZE      ; 0782
                                              7D  1B 0001F          BLEQU   4$
                      50              16      A7  9A 00021          MOVZBL  22(IDX_DFN), R0         ; 0796
                              50              09  78 00025          ASHL    #9, R0, R0
                              50            FC A045 9E 00029        MOVAB   -4(R0)[BKT_ADDR], OLD_END
                              51              60  3C 0002E          MOVZWL  (OLD_END), R1           ; 0797
                              51              55  C0 00031          ADDL2   BKT_ADDR, R1
                      51      50              51  C3 00034          SUBL3   R1, OLD_END, R1
                              51              52  C6 00038          DIVL2   OLD_SIZE, NUM_RECS
                      53      51              6E  C5 0003B          MULL3   NEW_SIZE, NUM_RECS, R3  ; 0798
                      54      50              53  C3 0003F          SUBL3   R3, OLD_END, NEW_END
                                        0C    A5  95 00043          TSTB    12(BKT_ADDR)            ; 0800
                                              07  12 00046          BNEQ    1$
                              03              1C  A7  E9 00048          BLBC    28(IDX_DFN), 1$     ; 0801
                              54              04  C2 0004C          SUBL2   #4, NEW_END             ; 0803
                      53      54              55  C3 0004F 1$:      SUBL3   BKT_ADDR, NEW_END, R3   ; 0805
                      60      53              01  A3 00053          SUBW3   #1, R3, (OLD_END)
                                    08      AE  0094  C9 00057          MOVL    148(IRAB), SAVE     ; 0809
                                  0094      C9    FF  A1 0005D          MOVAB   -1(R1), 148(IRAB)   ; 0810
                              53          0094  C9  9E 00063          MOVAB   148(R9), R3             ; 0812
                      04      AE              6E  78 00068          ASHL    #3, NEW_SIZE, 4(SP)     ; 0815
                              52              08  C4 0006D          MULL2   #8, R2
                                              63  D5 00070 2$:      TSTL    (R3)                    ; 0812
                                              1A  19 00072          BLSS    3$
                                          0000G 30 00074          BSBW    RM$CNTRL_ADDR           ; 0815
                      51      60              52  00  EF 00077          EXTZV   #0, R2, (R0), R1
                      64      04      AE      00  51  F0 0007C          INSV    R1, #0, 4(SP), (NEW_END)
                                              53  0094  C9 9E 00082          MOVAB   148(R9), R3     ; 0816
                                              63  D7 00087          DECL    (R3)
                                              54  6E  C0 00089          ADDL2   NEW_SIZE, NEW_END   ; 0817
                                              E2  11 0008C          BRB     2$
                                    0094      C9    08  AE  D0 0008E 3$:      MOVL    SAVE, 148(IRAB) ; 0820
                              50              6E  02  C3 00094          SUBL3   #2, NEW_SIZE, R0    ; 0824
                      0D      A5      02      03  50  F0 00098          INSV    R0, #3, #2, 13(BKT_ADDR)
                                          0000G 30 0009E 4$:      BSBW    RM$CNTRL_ADDR           ; 0835
                              51              50  D0 000A1          MOVL    R0, ADDR
                              50              8E  03  78 000A4          ASHL    #3, NEW_SIZE, R0    ; 0836
                      61      50              00    20  AE  F0 000A8          INSV    VBN, #0, R0, (ADDR)
                              5E              08  C0 000AE          ADDL2   #8, SP                  ; 0840
                                091C   8F  BA 000B1          POPR    #^M<R2,R3,R4,R8,R11>
                                           05 000B5          RSB
```

; Routine Size:  182 bytes,    Routine Base:  RM$RMS3 + 00D3

```
;  781          0841  1 %SBTTL  'RM$BKT_SORT'
;  782          0842  1 GLOBAL ROUTINE RM$BKT_SORT (BKT_ADDR) : RL$RABREG_7 =
;  783          0843  1
;  784          0844  1 !++
;  785          0845  1 !
;  786          0846  1 ! FUNCTIONAL DESCRIPTION:
;  787          0847  1 !
;  788          0848  1 ! this routine scans the bucket, makes a table of all used ID's,
;  789          0849  1 ! and then updates the NXTRECID and LSTRECID fields to reflect the existence
;  790          0850  1 ! of the lowest range of un-used ID's available.
;  791          0851  1 !
;  792          0852  1 ! CALLING SEQUENCE:
;  793          0853  1 !       RM$BKT_SORT (BKT_ADDR)
;  794          0854  1 !
;  795          0855  1 ! INPUT PARAMETERS:
;  796          0856  1 !       BKT_ADDR - pointer to bucket to be searched for ID's
;  797          0857  1 !
;  798          0858  1 ! IMPLICIT INPUTS:
;  799          0859  1 !       IDX_DFN - pointer to index descriptor
;  800          0860  1 !       IRAB - pointer to internal RAB
;  801          0861  1 !       IFAB - IFAB
;  802          0862  1 !
;  803          0863  1 ! OUTPUT PARAMETERS:
;  804          0864  1 !       NONE
;  805          0865  1 !
;  806          0866  1 ! IMPLICIT OUTPUTS:
;  807          0867  1 !       NONE
;  808          0868  1 !
;  809          0869  1 ! ROUTINE VALUE:
;  810          0870  1 !       low bit set if success (at least one ID was found)
;  811          0871  1 !       low bit clear if all ID's in use
;  812          0872  1 !
;  813          0873  1 ! SIDE EFFECTS:
;  814          0874  1 !       NXTRECID is set to lowest ID available
;  815          0875  1 !       LSTRECID is set to highest contiguous ID available after NXTRECID
;  816          0876  1 !
;  817          0877  1 !--
;  818          0878  1
;  819          0879  2     BEGIN
;  820          0880  2
;  821          0881  2     MAP
;  822          0882  2         BKT_ADDR        : REF BBLOCK;
;  823          0883  2
;  824          0884  2     EXTERNAL REGISTER
;  825          0885  2         R_IDX_DFN,
;  826          0886  2         R_IFAB,
;  827          0887  2         R_IRAB;
;  828          0888  2
;  829          0889  2     GLOBAL REGISTER
;  830          0890  2         R_REC_ADDR_STR;
;  831          0891  2
;  832          0892  2     LOCAL
;  833          0893  2         BIT_NUM,
;  834          0894  2         BIT_LEN;
;  835          0895  2
;  836          0896  2     ! create 8 longword array for table of used ID's
;  837          0897  2     !
```

```
;  838     0898  2
;  839     0899  2      STACKLOCAL
;  840     0900  2          USED_IDS        : VECTOR [8];
;  841     0901  2
;  842     0902  2      LOCAL
;  843     0903  2          EOB;
;  844     0904  2
;  845     0905  2      DECR I FROM 7 TO 0 DO
;  846     0906  2          USED_IDS[.I] = 0;
;  847     0907  2
;  848     0908  3      BEGIN
;  849     0909  3
;  850     0910  3      MAP
;  851     0911  3          USED_IDS        : BITVECTOR [256];
;  852     0912  3
;  853     0913  3      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  854     0914  3      EOB = .BKT_ADDR[BKT$W_FREESPACE] + .BKT_ADDR;
;  855     0915  3
;  856     0916  3      ! scan records in bucket.  set bit in bitvector for those id values found.
;  857     0917  3      !
;  858     0918  3      WHILE .REC_ADDR LSSA .EOB
;  859     0919  3      DO
;  860     0920  4          BEGIN
;  861     0921  4
;  862     0922  4          GLOBAL REGISTER
;  863     0923  4              R_RAB,
;  864     0924  4              R_IMPURE;
;  865     0925  4
;  866     0926  4          USED_IDS[.REC_ADDR[IRC$B_ID]] = 1;
;  867     0927  4          RM$GETNEXT_PEC();                                                    ; R
;  868     0928  3          END;
;  869     0929  3
;  870     0930  3      USED_IDS[0] = 0;                            ! flag set when a free id found
;  871     0931  3      USED_IDS[255] = 1;                          ! make sure last one always set
;  872     0932  3      END;                                        ! of USED_IDS as bitvector
;  873     0933  2      BIT_LEN = 31;
;  874     0934  2      BIT_NUM = 1;
;  875     0935  2
;  876     0936  2      ! now search the bit array for the first clear bit (that corresponds to
;  877     0937  2      ! an unused id value) one longword at a time.  when found, check the
;  878     0938  2      ! remaining bits in that longword looking for a used id.  if none found
;  879     0939  2      ! there, continue searching the remaining longwords in the array to
;  880     0940  2      ! determine the extent of the unused id values.
;  881     0941  2      !
;  882     0942  2      INCR I FROM 0 TO 7 DO
;  883     0943  2
;  884     0944  2          IF NOT .USED_IDS
;  885     0945  2          THEN
;  886     0946  3              BEGIN
;  887     0947  3
;  888     0948  3              IF NOT FFC(BIT_NUM, BIT_LEN, USED_IDS[.I], BIT_NUM)
;  889     0949  3              THEN
;  890     0950  4                  BEGIN
;  891     0951  4                  BKT_ADDR[BKT$B_NXTRECID] = .I*32 + .BIT_NUM;
;  892     0952  4                  USED_IDS<0, 1> = 1;
;  893     0953  4                  BIT_LEN = 32 - .BIT_NUM;
;  894     0954  4
```

```
;  895          0955  4                  IF NOT FFS(BIT_NUM, BIT_LEN, USED_IDS[.I], BIT_NUM)
;  896          0956  4                  THEN
;  897          0957  5                      BEGIN
;  898          0958  5                      BKT_ADDR[BKT$B_LSTRECID] = .I*32 + .BIT_NUM - 1;
;  899          0959  5                      EXITLOOP;
;  900          0960  5
;  901          0961  5                      END
;  902          0962  4                  ELSE
;  903          0963  4                      BIT_LEN = 32;
;  904          0964  4
;  905          0965  3                  END;
;  906          0966  3
;  907          0967  3              BIT_NUM = 0;
;  908          0968  3              END
;  909          0969  2          ELSE
;  910          0970  2
;  911          0971  2              IF NOT FFS(BIT_NUM, BIT_LEN, USED_IDS[.I], BIT_NUM)
;  912          0972  2              THEN
;  913          0973  3                  BEGIN
;  914          0974  3                  BKT_ADDR[BKT$B_LSTRECID] = .I*32 + .BIT_NUM - 1;
;  915          0975  3                  EXITLOOP;
;  916          0976  3
;  917          0977  3                  END
;  918          0978  2              ELSE
;  919          0979  3                  BEGIN
;  920          0980  3                  BIT_LEN = 32;
;  921          0981  3                  BIT_NUM = 0;
;  922          0982  2                  END;
;  923          0983  2
;  924          0984  2      ! if this is clear, no free id's were found.  if set, at least one
;  925          0985  2      ! is available.
;  926          0986  2      !
;  927          0987  2      RETURN .USED_IDS[0];
;  928          0988  2
;  929          0989  1      END;


                 094C   8F   BB 00000 RM$BKT_SORT::
                                      PUSHR   #^M<R2,R3,R6,R8,R11>                    ; 0842
                     5E        20   C2 00004                SUBL2   #32, SP
                     50        07   D0 00007                MOVL    #7, I              ; 0905
                   6E40        D4   D4 0000A 1$:            CLRL    USED_IDS[I]        ; 0906
                     FA        50   F4 0000D                SOBGEQ  I, 1$
                     52   38   AE   D0 00010                MOVL    BKT_ADDR, R2       ; 0913
                     56   0E   A2   9E 00014                MOVAB   14(R2), REC_ADDR
                     53   04   A2   3C 00018                MOVZWL  4(R2), EOB         ; 0914
                     53        52   C0 0001C                ADDL2   R2, EOB
                     53        56   D1 0001F 2$:            CMPL    REC_ADDR, EOB      ; 0918
                               0D   1E 00022                BGEQU   4$
                     50   01   A6   9A 00024                MOVZBL  1(REC_ADDR), R0    ; 0926
                 00  6E        50   E2 00028                BBSS    R0, USED_IDS, 3$
                           0000G  30 0002C 3$:              BSBW    RM$GETNEXT_REC     ; 0927
                               EE   11 0002F                BRB     2$                 ; 0918
                     6E        01   8A 00031 4$:            BICB2   #1, USED_IDS       ; 0930
```

```
                                 1F   AE        80    8F   88 00034          BISB2   #128, USED_IDS+31                                  ; 0931
                                      58              1F   D0 00039          MOVL    #31, BIT_LEN                                       ; 0933
                                      56              01   D0 0003C          MOVL    #1, BIT_NUM                                        ; 0934
                                                      50   D4 0003F          CLRL    I                                                  ; 0942
                                      5B            6E40   DE 00041 5$:       MOVAL   USED_IDS[I], R11                                   ; 0948
                                      20              6E   E8 00045          BLBS    USED_IDS, 6$                                       ; 0944
                 56                         6B          58   EB 00048          FFC     BIT_NUM, BIT_LEN, (R11), BIT_NUM                   ; 0948
                                                         32   13 0004D          BEQL    9$                                                 ;
                                            51           50   78 0004F          ASHL    #5, I, R1                                          ; 951
                                  06        A2           51   81 00053          ADDB3   BIT_NUM, R1, 6(R2)                                 ;
                                            6E           01   88 00058          BISB2   #1, USED_IDS                                       ; 0952
                                            58           20   C3 0005B          SUBL3   BIT_NUM, #32, BIT_LEN                              ; 0953
                 56                         6B          58   EA 0005F          FFS     BIT_NUM, BIT_LEN, (R11), BIT_NUM                   ; 0955
                                                         18   13 00064          BEQL    8$                                                 ;
                                                         0B   11 00066          BRB     7$                                                 ; 0958
                 56                         6B          58   EA 00068 6$:       FFS     BIT_NUM, BIT_LEN, (R11), BIT_NUM                   ; 0971
                                                         0F   13 0006D          BEQL    8$                                                 ;
                                            51           50   78 0006F          ASHL    #5, I, R1                                          ; 0974
                                                         53   9E 00073 7$:       MOVAB   -1(BIT_NUM)[R1], R3                                ;
                                  07        A2           53   90 00078          MOVB    R3, 7(R2)                                          ;
                                                         09   11 0007C          BRB     10$                                                ; 0973
                                            58           20   D0 0007E 8$:       MOVL    #32, BIT_LEN                                       ; 0980
                                                         56   D4 00081 9$:       CLRL    BIT_NUM                                            ; 0981
                                            BA           50   F3 00083          AOBLEQ  #7, I, 5$                                          ; 0944
                                                         50   8E   D0 00087 10$:      MOVL    USED_IDS, R0                               ; 0987
                                                         5E    C   C0 0008A          ADDL2   #28, SP                                    ; 0989
                                                  094C    8   BA 0008D          POPR    #^M<R2,R3,R6,R8,R11>                               ;
                                                              05 00091          RSB                                                        ;

; Routine Size:  146 bytes,    Routine Base:  RM$RMS3 + 0189
```

```
;  921          0990  1 %SBTTL  'RM$BLD_IDX_REC'
;  932          0991  1 ROUTINE RM$BLD_IDX_REC (REC_SZ) : RL$COMMON_LINK NOVALUE =
;  933          0992  1
;  934          0993  1 !++
;  935          0994  1 !
;  936          0995  1 ! FUNCTIONAL DESCRIPTION:
;  937          0996  1 !
;  938          0997  1 ! This routine builds an index record in the buffer.  It will do one of the
;  939          0998  1 ! following:
;  940          0999  1 !
;  941          1000  1 !       0) Call RM$V3_IDX_REC to do everything for prologue three index
;  942          1001  1 !       buckets.
;  943          1002  1 !
;  944          1003  1 !       1) replace key value of existing down pointer with the new high key
;  945          1004  1 !       value (in keybuffer 2) of the original bucket (described as vbn_left)
;  946          1005  1 !       and use that key value along with the value of vbn_right to create
;  947          1006  1 !       a new index record pointing to the right hand bucket of the split.
;  948          1007  1 !
;  949          1008  1 !       2) same as the above except create another record between the two
;  950          1009  1 !       described using the contents of keybuffer 3 and vbn_right.  this
;  951          1010  1 !       record points to the middle record of a 3 or 4 bucket split.
;  952          1011  1 !
;  953          1012  1 !       3) as an option on the first situation (1), change the down
;  954          1013  1 !       pointer to be the value of vbn_mid if the current down pointer matches
;  955          1014  1 !       the value in vbn_left.
;  956          1015  1 !
;  957          1016  1 !       4) build only the left hand part of the possible pieces described above
;  958          1017  1 !       and make the break just before the potential new middle bucket pointer.
;  959          1018  1 !
;  960          1019  1 !       5) build the part remaining after only doing 4).
;  961          1020  1 !
;  962          1021  1 ! Any modifications to this code should be done only after thoroughly
;  963          1022  1 ! understanding the interactions between this routine, RM$RECORD_SIZE,
;  964          1023  1 ! and RM$SPLIT_EM.
;  965          1024  1 !
;  966          1025  1 ! CALLING SEQUENCE:
;  967          1026  1 !       RM$BLD_IDX_REC (REC_SZ)
;  968          1027  1 !
;  969          1028  1 ! INPUT PARAMETERS:
;  970          1029  1 !       REC_SZ  - total size of record to be inserted
;  971          1030  1 !
;  972          1031  1 ! IMPLICIT INPUTS:
;  973          1032  1 !       REC_ADDR - points to location in buffer where record is to be built
;  974          1033  1 !
;  975          1034  1 !       IRAB    [vbn_left] - left hand (original) bucket in split.  zero if
;  976          1035  1 !                            only right hand part of record to be built.
;  977          1036  1 !               [vbn_mid] - possible middle bucket in 3 or 4 bucket split
;  978          1037  1 !               [vbn_right] - right hand bucket of split
;  979          1038  1 !               [empty_bkt] - left hand bucket is dead, i.e., no data records
;  980          1039  1 !               [big_split] - more than 1 new bucket in split
;  981          1040  1 !               [spl_idx] - only build left hand part of index record
;  982          1041  1 !               [pos_ins] - position of insert of the new record
;  983          1042  1 !               [keybuf] - address of keybuffers
;  984          1043  1 !
;  985          1044  1 !       IFAB    [kbufsz] - size of keybuffers
;  986          1045  1 !
;  987          1046  1 !       IDX_DFN [keysz] - size of this key
```

```
;  988          1047  1 !
;  989          1048  1 ! OUTPUT PARAMETERS:
;  990          1049  1 !       NONE
;  991          1050  1 !
;  992          1051  1 ! IMPLICIT OUTPUTS:
;  993          1052  1 !       REC_ADDR - points to 'irst byte beyond record.
;  994          1053  1 !
;  995          1054  1 ! ROUTINE VALUE:
;  996          1055  1 !       NONE
;  997          1056  1 !
;  998          1057  1 ! SIDE EFFECTS:
;  999          1058  1 !       NONE
; 1000          1059  1 !
; 1001          1060  1 !--
; 1002          1061  1
; 1003          1062  2     BEGIN
; 1004          1063  2
; 1005          1064  2     EXTERNAL REGISTER
; 1006          1065  2         R_BKT_ADDR_STR,
; 1007          1066  2         R_REC_ADDR_STR,
; 1008          1067  2         R_IRAB_STR,
; 1009          1068  2         R_IDX_DFN_STR,
; 1010          1069  2         R_IFAB_STR;
; 1011          1070  2
; 1012          1071  2     GLOBAL REGISTER
; 1013          1072  2         R_BDB,
; 1014          1073  2         R_IMPURE,
; 1015          1074  2         R_RAB;
; 1016          1075  2
; 1017          1076  2     IF .IFAB [IFB$B_PLG_VER] EQLU PLG$C_VER_3
; 1018          1077  2     THEN
; 1019          1078  2
; 1020          1079  2         ! Prologue three index records. They basically
; 1021          1080  2         ! have nothing in common with pre-prologue three
; 1022          1081  2         ! index records. Just call the routine that
; 1023          1082  2         ! handles them and get out.
; 1024          1083  2         !
; 1025          1084  3         BEGIN
; 1026          1085  3
; 1027          1086  3         RM$V3_IDX_REC (.REC_SZ);
; 1028          1087  3         RETURN
; 1029          1088  2         END;
; 1030          1089  2
; 1031          1090  2     ! first key we have to build if present is the new key for the left
; 1032          1091  2     ! hand bucket this will be zero only when splitting up index record when
; 1033          1092  2     ! two new key values are required - currently 'empty' buckets (i.e.,
; 1034          1093  2     ! buckets which end up without any records in them after a split) are not
; 1035          1094  2     ! removed from the index
; 1036          1095  2     !
; 1037          1096  2
; 1038          1097  2     IF .IRAB[IRB$L_VBN_LEFT] NEQ 0
; 1039          1098  2     THEN
; 1040          1099  2
; 1041          1100  2         ! the bucket has already been spread apart but the index record we
; 1042          1101  2         ! positioned to still exists where we found it, so using the existing
; 1043          1102  2         ! pointer, change the key value to the high key of VBN_LEFT
; 1044          1103  2         !
```

```
;1045     1104  3          BEGIN
;1046     1105  3
;1047     1106  3          ! check for empty bucket situation, compare down pointers before
;1048     1107  3          ! deciding whether to modify down pointer or not
;1049     1108  3          !
;1050     1109  3
;1051     1110  3          IF .IRAB[IRB$V_EMPTY_BKT]
;1052     1111  3          THEN
;1053     1112  4              BEGIN
;1054     1113  4
;1055     1114  4              BUILTIN
;1056     1115  4                  AP;
;1057     1116  4
;1058     1117  4              AP = 1;                             ! set for index level
;1059     1118  4
;1060     1119  4              IF .IRAB[IRB$L_VBN_LEFT] EQLA RM$RECORD_VBN()
;1061     1120  4
;1062     1121  4                  ! current down pointer matches so modify down pointer
;1063     1122  4                  !
;1064     1123  4              THEN
;1065     1124  4                  RM$MOVE_IN_VBN(.IRAB[IRB$L_VBN_MID])
;1066     1125  4              ELSE
;1067     1126  4                  REC_ADDR = .REC_ADDR + .REC_ADDR[IRC$V_PTRSZ] + IRC$C_IDXPTRBAS
;1068     1127  4                  + IRC$C_IDXOVHDSZ;              ! advance past pointer
;1069     1128  4
;1070     1129  4              IF NOT .IRAB[IRB$V_BIG_SPLIT]
;1071     1130  4              THEN
;1072     1131  4                  RETURN;                         ! no more to do, quit
;1073     1132  4
;1074     1133  4              END
;1075     1134  3          ELSE
;1076     1135  3              REC_ADDR = .REC_ADDR + .REC_ADDR[IRC$V_PTRSZ] + IRC$C_IDXPTRBAS +
;1077     1136  3              IRC$C_IDXOVHDSZ;
;1078     1137  3
;1079     1138  3          ! move in key from keybuffer 2
;1080     1139  3          !
;1081     1140  3          REC_ADDR = RM$MOVE(.IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(2), .REC_ADDR);
;1082     1141  3
;1083     1142  3          ! We move in the VBN from IRB$L_VBN_RIGHT if and only if this is a
;1084     1143  3          ! one-pass multi-bucket empty bucket split case.
;1085     1144  3          !
;1086     1145  4          IF (.IRAB[IRB$V_EMPTY_BKT]
;1087     1146  4              AND
;1088     1147  4             .IRAB[IRB$V_BIG_SPLIT]
;1089     1148  4              AND
;1090     1149  4             NOT .IRAB[IRB$V_SPL_IDX])
;1091     1150  3          THEN
;1092     1151  3              RM$MOVE_IN_VBN(.IRAB[IRB$L_VBN_RIGHT]);
;1093     1152  3
;1094     1153  3          ! We are done if this was a empty bucket split case
;1095     1154  3          !
;1096     1155  3          IF .IRAB[IRB$V_EMPTY_BKT]
;1097     1156  3          THEN
;1098     1157  3              RETURN;
;1099     1158  3
;1100     1159  2          END;
;1101     1160  2
```

```
;1102        1161  2      ! At this point all bucket splits involving empty buckets have completed
;1103        1162  2      ! except for the second pass of the two-pass multi-bucket with empty
;1104        1163  2      ! bucket split case.
;1105        1164  2      !
;1106        1165  2      ! We continue for all other cases with one exception. When the insertion
;1107        1166  2      ! point is found to be at the split point, two passes must be made - one
;1108        1167  2      ! to update the old bucket, and the second to update the new bucket. If
;1109        1168  2      ! this is the first pass to update the old bucket components we do not
;1110        1169  2      ! continue unless a multibucket split had occurred, two and only two
;1111        1170  2      ! index records could fit in an index bucket, and the split point requires
;1112        1171  2      ! both of them to be moved to the new bucket to make room for the two new
;1113        1172  2      ! index records. In all other "first pass" situations we do not continue.
;1114        1173  2      !
;1115        1174  2      IF NOT .IRAB[IRB$V_SPL_IDX]
;1116        1175  2          OR
;1117        1176  2         (.IRAB[IRB$V_SPL_IDX]
;1118        1177  3              AND .IRAB[IRB$V_BIG_SPLIT]
;1119        1178  3              AND .IRAB[IRB$W_POS_INS] EQLU BKT$C_OVERHDSZ)
;1120        1179  2      THEN
;1121        1180  3          BEGIN
;1122        1181  3
;1123        1182  3          ! If a multibucket split occurred, the middle VBN and its corresponding
;1124        1183  3          ! key are inserted into the current index bucket with two exceptions.
;1125        1184  3          ! If this is the second pass of a two-pass multibucket split case where
;1126        1185  3          ! both records had to be put in the old index bucket, or of a two-pass
;1127        1186  3          ! multi-bucket empty bucket split case, then this bucket updating does
;1128        1187  3          ! not take place.
;1129        1188  3          !
;1130        1189  3          IF .IRAB[IRB$V_BIG_SPLIT]
;1131        1190  3              AND
;1132        1191  4             (.IRAB[IRB$L_VBN_MID] NEQU 0)
;1133        1192  3          THEN
;1134        1193  4              BEGIN
;1135        1194  4              RM$MOVE_IN_VBN(.IRAB[IRB$L_VBN_MID]);
;1136        1195  4
;1137        1196  4              ! move in key from keybuffer 3
;1138        1197  4              !
;1139        1198  4              REC_ADDR = RM$MOVE(.IDX_DFN[IDX$B_KEYSZ],
;1140        1199  4                  KEYBUF_ADDR(3), .REC_ADDR);
;1141        1200  3              END;
;1142        1201  3
;1143        1202  3          ! Unless this is the first pass of one of the three two-pass index
;1144        1203  3          ! bucket split cases, we always swing the pointer for VBN right.
;1145        1204  3          ! If this is the first of two passes, we do not swing the pointer
;1146        1205  3          ! because this pass is used to update the old index bucket components
;1147        1206  3          ! and VBN_RIGHT must be swung during the second pass when the new
;1148        1207  3          ! index components are updated.
;1149        1208  3          !
;1150        1209  3          IF NOT .IRAB[IRB$V_SPL_IDX]
;1151        1210  3          THEN
;1152        1211  3              RM$MOVE_IN_VBN(.IRAB[IRB$L_VBN_RIGHT]);
;1153        1212  2          END;
;1154        1213  2
;1155        1214  1      END;
```

```
                        0914   8F  BB 00000 RM$BLD_IDX_REC:
                                                PUSHR   #^M<R2,R4,R8,R11>                       ; 0991
                    03  00B7   CA  91 00004        CMPB    183(IFAB), #3                       ; 1076
                            0B  12 00009        BNEQ    1$                                      ;
                          14    AE  DD 0000B        PUSHL   REC_SZ                              ; 1086
                        0000V   30  30 0000E        BSBW    RM$V3_IDX_REC                       ;
                    5E              04  C0 00011        ADDL2   #4, SP                          ;
                                    30  11 00014        BRB     4$                              ; 1084
                    51  0088        C9  D0 00016 1$:     MOVL    136(IRAB), R1                   ; 1097
                                    69  13 0001B        BEQL    8$                              ;
                27      44  A9              06  E1 0001D        BBC     #6, 68(IRAB), 5$        ; 1110
                    5C              01  D0 00022        MOVL    #1, AP                          ; 1117
                                00C0G   30  30 00025        BSBW    RM$RECORD_VBN               ; 1119
                    50              51  D1 00028        CMPL    R1, R0                          ;
                                    0A  12 0002B        BNEQ    2$                              ;
                    52  0090        C9  D0 0002D        MOVL    144(IRAB), R2                   ; 1124
                                0000V   30  30 00032        BSBW    RM$MOVE_IN_VBN              ;
                                    0A  11 00035        BRB     3$                              ;
        50          66              02  EF 00037 2$:     EXTZV   #0, #2, (REC_ADDR), R0          ; 1126
                    56    03    A046  9E 0003C        MOVAB   3(R0)[REC_ADDR], REC_ADDR       ; 1127
                0D      44  A9              02  E0 00041 3$:     BBS     #2, 68(IRAB), 6$        ; 1129
                                0083    31  30 00046 4$:     BRW     11$                         ; 1131
        50          66              02  EF 00049 5$:     EXTZV   #0, #2, (REC_ADDR), R0          ; 1135
                    56    03    A046  9E 0004E        MOVAB   3(R0)[REC_ADDR], REC_ADDR       ;
                              56    DD 00053 6$:     PUSHL   REC_ADDR                        ; 1140
                    50  00B4        CA  3C 00055        MOVZWL  180(IFAB), R0                   ;
                          60  B940  9F  0005A        PUSHAB  @96(IRAB)[R0]                   ;
                    7E      20    A7  9A 0005E        MOVZBL  32(IDX_DFN), -(SP)              ;
                                00C0G   30  30 00062        BSBW    RM$MOVE                     ;
                    5E              0C  C0 00065        ADDL2   #12, SP                         ;
                    56              50  D0 00068        MOVL    R0, REC_ADDR                    ;
            16      44  A9              06  E1 0006B        BBC     #6, 68(IRAB), 8$        ; 1145
            0C      44  A9              02  E1 00070        BBC     #2, 68(IRAB), 7$        ; 1147
                        08      44  A9  E8 00075        BLBS    68(IRAB), 7$                ; 1149
                    52      008C    C9  D0 00079        MOVL    140(IRAB), R2               ; 1151
                                0000V   30  30 0007E        BSBW    RM$MOVE_IN_VBN              ;
            46      44  A9              06  E0 00081 7$:     BBS     #6, 68(IRAB), 11$       ; 1155
                        0B      44  A9  E9 00086 8$:     BLBC    68(IRAB), 9$                ; 1174
            3D      44  A9              02  E1 0008A        BBC     #2, 68(IRAB), 11$       ; 1177
                        0E      48  A9  B1 0008F        CMPW    72(IRAB), #14               ; 1178
                                    37  12 00093        BNEQ    11$                         ;
            26      44  A9              02  E1 00095 9$:     BBC     #2, 68(IRAB), 10$       ; 1189
                                0090    C9  D5 0009A        TSTL    144(IRAB)                   ; 1191
                                    20  13 0009E        BEQL    10$                         ;
                    52  0090        C9  D0 000A0        MOVL    144(IRAB), R2               ; 1194
                                0000V   30  30 000A5        BSBW    RM$MOVE_IN_VBN              ;
                              56    DD 000A8        PUSHL   REC_ADDR                        ; 1199
                    50  00B4        CA  3C 000AA        MOVZWL  180(IFAB), R0                   ;
                          60  B940  3F  000AF        PUSHAW  @96(IRAB)[R0]                   ;
                    7E      20    A7  9A 000B3        MOVZBL  32(IDX_DFN), -(SP)              ; 1198
                                00C0G   30  30 000B7        BSBW    RM$MOVE                     ;
                    5E              0C  C0 000BA        ADDL2   #12, SP                         ;
                    56              50  D0 000BD        MOVL    R0, REC_ADDR                    ;
                        08      44  A9  E8 000C0 10$:    BLBS    68(IRAB), 11$               ; 1209
                    52      008C    C9  D0 000C4        MOVL    140(IRAB), R2               ; 1211
```

```
                     0000V 30 000C9          BSBW    RM$MOVE_IN_VBN                  ;
                0914   8F  BA 000CC 11$:     POPR    #^M<R2,R4,R8,R11>               ; 1214
                           05 000D0          RSB                                     ;

; Routine Size:  209 bytes,    Routine Base:  RM$RMS3 + 021B
```

```
;1157     1215  1 %SBTTL  'RM$BLD_NEW_SIDR'
;1158     1216  1 ROUTINE RM$BLD_NEW_SIDR (SIDR_SIZE) : RL$COMMON_LINK NOVALUE =
;1159     1217  1
;1160     1218  1 !++
;1161     1219  1 !
;1162     1220  1 ! FUNCTIONAL DESCRIPTION:
;1163     1221  1 !
;1164     1222  1 !       This routine builds a new SIDR using the key found in keybuffer 2.
;1165     1223  1 !
;1166     1224  1 ! CALLING SEQUENCE:
;1167     1225  1 !
;1168     1226  1 !       RM$BLD_NEW_SIDR()
;1169     1227  1 !
;1170     1228  1 ! INPUT PARAMETERS:
;1171     1229  1 !
;1172     1230  1 !       SIDR_SIZE       - full size of the new SIDR
;1173     1231  1 !
;1174     1232  1 ! IMPLICIT INPUTS:
;1175     1233  1 !
;1176     1234  1 !       BKT_ADDR                - address of SIDR bucket
;1177     1235  1 !           BKT$W_FREESPACE     - offset to first free byte in bucket
;1178     1236  1 !           BKT$B_NXTRECID      - next available ID in bucket
;1179     1237  1 !
;1180     1238  1 !       IDX_DFN                 - address of index descriptor
;1181     1239  1 !           IDX$V_DUPKEYS       - if set, duplicate alternate keys are allowed
;1182     1240  1 !           IDX$V_KEY_COMPR     - if set, SIDR key compression is enabled
;1183     1241  1 !           IDX$B_KEYSZ         - size of alternate key
;1184     1242  1 !
;1185     1243  1 !       IFAB                    - addres of IFAB
;1186     1244  1 !           IFB$W_KBUFSZ        - size of each of the contigious keybuffers
;1187     1245  1 !           IFB$B_PLG_VER       - prologue version of file
;1188     1246  1 !
;1189     1247  1 !       IRAB                    - address of IRAB
;1190     1248  1 !           IRB$V_CONT_BKT      - if set, bucket is a continuation bucket
;1191     1249  1 !           IRB$V_DUP_KEY       - if set, not first SIDR with this key value
;1192     1250  1 !           IRB$L_KEYBUF        - address of contigious keybuffers
;1193     1251  1 !
;1194     1252  1 !       REC_ADDR                - address of point of insertion of new SIDR
;1195     1253  1 !
;1196     1254  1 ! OUTPUT PARAMETERS:
;1197     1255  1 !       NONE
;1198     1256  1 !
;1199     1257  1 ! IMPLICIT OUTPUTS:
;1200     1258  1 !
;1201     1259  1 !       IRAB[IRB$L_LST_REC]     - address of new SIDR
;1202     1260  1 !       REC_ADDR                - address of first byte following new SIDR
;1203     1261  1 !
;1204     1262  1 ! ROUTINE VALUE:
;1205     1263  1 !       NONE
;1206     1264  1 !
;1207     1265  1 ! SIDE EFFECTS:
;1208     1266  1 !
;1209     1267  1 !       If the file is a prologue 1 or 2 file, BKT_ADDR[BKT$B_NXTRECID] will
;1210     1268  1 !           have been incremented by 1.
;1211     1269  1 !       If key compression is enabled, and a SIDR follows the new SIDR being
;1212     1270  1 !           added, then its key will be re-compressed, and the SIDR bucket's
;1213     1271  1 !           freespace offset pointer updated appropriately.
```

```
;1214     1272  1 !
;1215     1273  1 !--
;1216     1274  1
;1217     1275  2     BEGIN
;1218     1276  2
;1219     1277  2     EXTERNAL REGISTER
;1220     1278  2         R_BKT_ADDR_STR,
;1221     1279  2         R_IFAB_STR,
;1222     1280  2         R_IRAB_STR,
;1223     1281  2         R_IDX_DFN_STR,
;1224     1282  2         R_REC_ADDR_STR;
;1225     1283  2
;1226     1284  2     LOCAL
;1227     1285  2         FIRST_ARRAY_ADDR        : REF BBLOCK;
;1228     1286  2
;1229     1287  2     ! Save the address of the point of insertion of the new SIDR.
;1230     1288  2     !
;1231     1289  2     IRAB[IRB$L_LST_REC] = .REC_ADDR;
;1232     1290  2
;1233     1291  2     ! Create the record control byte, DUP_CNT (if appropriate), and record ID
;1234     1292  2     ! fields for the prologue 1 or 2 SIDR. It is not necessary to do this
;1235     1293  2     ! for prologue 3 SIDRs because they don't have any of these fields. After
;1236     1294  2     ! this step, REC_ADDR will be pointing to the size field of the new SIDR,
;1237     1295  2     ! regardless of the file's prologue version.
;1238     1296  2     !
;1239     1297  2     IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
;1240     1298  2     THEN
;1241     1299  3         BEGIN
;1242     1300  3
;1243     1301  3         ! If duplicate keys are allowed on this index, and this bucket is not a
;1244     1302  3         ! continuation bucket, then a DUP_CNT field will be created as part
;1245     1303  3         ! of the overhead of the new SIDR even though this field will not be
;1246     1304  3         ! maintained. The control byte, ID, and DUP_CNT fields are initialized.
;1247     1305  3         !
;1248     1306  3         IF .IDX_DFN[IDX$V_DUPKEYS]
;1249     1307  3             AND
;1250     1308  3            NOT .IRAB[IRB$V_CONT_BKT]
;1251     1309  3         THEN
;1252     1310  4             BEGIN
;1253     1311  4             (.REC_ADDR)<0, 8> = 1;
;1254     1312  4             REC_ADDR = .REC_ADDR + 1;
;1255     1313  4
;1256     1314  4             (.REC_ADDR)<0, 8> = .BKT_ADDR[BKT$B_NXTRECID];
;1257     1315  4             REC_ADDR = .REC_ADDR + 1;
;1258     1316  4
;1259     1317  4             (.REC_ADDR)<0, 32> = 0;
;1260     1318  4             REC_ADDR = .REC_ADDR + IRC$C_DCNTSZFLD;
;1261     1319  4             END
;1262     1320  4
;1263     1321  4         ! Either duplicate alternate keys are not allowed, or if they are, this
;1264     1322  4         ! bucket is a continuation bucket. In either case, it is not necessary
;1265     1323  4         ! to allocate and initialize a DUP_CNT field. Just the control byte
;1266     1324  4         ! and the ID field of the new record are intialized.
;1267     1325  4         !
;1268     1326  3         ELSE
;1269     1327  4             BEGIN
;1270     1328  4             (.REC_ADDR)<0, 8> = IRC$M_NODUPCNT;
```

```
;1271     1329  4              REC_ADDR = .REC_ADDR + 1;
;1272     1330  4
;1273     1331  4              (.REC_ADDR)<0, 8> = .BKT_ADDR[BKT$B_NXTRECID];
;1274     1332  4              REC_ADDR = .REC_ADDR + 1;
;1275     1333  3              END;
;1276     1334  3
;1277     1335  3          ! Increment the next record ID field in the bucket.
;1278     1336  3          !
;1279     1337  3          BKT_ADDR[BKT$B_NXTRECID] = .BKT_ADDR[BKT$B_NXTRECID] + 1;
;1280     1338  2          END;
;1281     1339  2
;1282     1340  2      ! If key compression is not enabled, copy the new SIDR's key from keybuffer
;1283     1341  2      ! 2 into its place in the SIDR bucket, and initialize the new SIDR's record
;1284     1342  2      ! size to be the size of the key. This size field will be updated to its
;1285     1343  2      ! correct value (key size + SIDR array size) by the routine which will
;1286     1344  2      ! create the first element in the SIDR array.
;1287     1345  2      !
;1288     1346  3      BEGIN
;1289     1347  3
;1290     1348  3      GLOBAL REGISTER
;1291     1349  3          R_BDB,
;1292     1350  3          R_IMPURE,
;1293     1351  3          R_RAB;
;1294     1352  3
;1295     1353  3      IF NOT .IDX_DFN[IDX$V_KEY_COMPR]
;1296     1354  3      THEN
;1297     1355  4          BEGIN
;1298     1356  4          (.REC_ADDR)<0, 16> = .IDX_DFN[IDX$B_KEYSZ];
;1299     1357  4          REC_ADDR = .REC_ADDR + IRC$C_DATSZFLD;
;1300     1358  4
;1301     1359  4          REC_ADDR = RM$MOVE(.IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(2), .REC_ADDR);
;1302     1360  4          END
;1303     1361  4
;1304     1362  4      ! If key compression is enabled, the key in keybuffer 2 must first be
;1305     1363  4      ! compressed before it is moved into the SIDR bucket. The key of the
;1306     1364  4      ! following SIDR (if there is one) must also be re-compressed using the
;1307     1365  4      ! new SIDR's key. Finally, the size field must be initialized to the size of
;1308     1366  4      ! the new SIDR's compressed key plus the two bytes of key compression
;1309     1367  4      ! overhead. This size field will be updated to its correct value (key size +
;1310     1368  4      ! SIDR array size) by the routine which will create the first element in
;1311     1369  4      ! the SIDR array.
;1312     1370  4      !
;1313     1371  3      ELSE
;1314     1372  4          BEGIN
;1315     1373  4
;1316     1374  4          LOCAL
;1317     1375  4              KEYBUF      : REF BBLOCK;
;1318     1376  4
;1319     1377  4          ! Compress the new SIDR's key within its keybuffer - keybuffer 2.
;1320     1378  4          !
;1321     1379  4          KEYBUF = KEYBUF_ADDR(2);
;1322     1380  4          RM$COMPRESS_KEY (.KEYBUF);
;1323     1381  4
;1324     1382  4          ! Fix up the front compression of the following record because it might
;1325     1383  4          ! change due to the insertion of this new SIDR. Of course, this step
;1326     1384  4          ! is unnecessary, if there is no following record.
;1327     1385  4          !
```

```
;  1328     1386  5          IF (.REC_ADDR + .SIDR_SIZE LSSU .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
;  1329     1387  4          THEN
;  1330     1388  4              RM$RECOMPR_KEY (.KEYBUF, .REC_ADDR + IRC$C_DATSZFLD + .SIDR_SIZE);
;  1331     1389  4
;  1332     1390  4          ! Initialize the size field of the new SIDR to the size of the
;  1333     1391  4          ! compressed key plus the two bytes of key compression overhead, and
;  1334     1392  4          ! move the new SIDR key into the bucket including the two bytes of
;  1335     1393  4          ! compression overhead.
;  1336     1394  4          !
;  1337     1395  4          (.REC_ADDR)<0,16> = .KEYBUF[KEY_LEN] + IRC$C_KEYCMPOVH;
;  1338     1396  4          REC_ADDR = .REC_ADDR + IRC$C_DATSZFLD;
;  1339     1397  4
;  1340     1398  4          REC_ADDR = RM$MOVE (.KEYBUF[KEY_LEN] + 2, .KEYBUF, .REC_ADDR);
;  1341     1399  3          END;
;  1342     1400  2      END;
;  1343     1401  2
;  1344     1402  2      ! Save the address of what will be the first array element in this SIDR
;  1345     1403  2      ! before creating the array.
;  1346     1404  2      !
;  1347     1405  2      FIRST_ARRAY_ADDR = .REC_ADDR;
;  1348     1406  2      RM$ADD_TO_ARRAY();
;  1349     1407  2
;  1350     1408  2      ! If this is a prologue 3 file, and this is the first such SIDR with this
;  1351     1409  2      ! key value in the file, then set a bit indicating this in the control byte
;  1352     1410  2      ! of the SIDR array's first element.
;  1353     1411  2      !
;  1354     1412  2      IF  .IFAB[IFB$B_PLG_VER] GEQU PLG$C_VER_3
;  1355     1413  2       AND
;  1356     1414  2          NOT .IRAB[IRB$V_DUP_KEY]
;  1357     1415  2      THEN
;  1358     1416  2          FIRST_ARRAY_ADDR[IRC$V_FIRST_KEY] = 1;
;  1359     1417  1      END;


                                0914   8F   BB 00000 RM$BLD_NEW_SIDR:
                                                     PUSHR   #^M<R2,R4,R8,R11>                ;  1216
                          4C    A9                   56   D0 00004          MOVL    REC_ADDR, 76(IRAB)       ;  1289
                          03          00B7           CA   91 00008          CMPB    183(IFAB), #3            ;  1297
                                                     1E   1E 0000D          BGEQU   3$                       ;
                          10          1C             A7   E9 0000F          BLBC    28(IDX_DFN), 1$          ;  1306
                    0B    44    A9                   04   E0 00013          BBS     #4, 68(IRAB), 1$         ;  1308
                          86                         01   90 00018          MOVB    #1, (REC_ADDR)+          ;  1311
                          86          06             A5   90 0001B          MOVB    6(BKT_ADDR), (REC_ADDR)+ ;  1314
                                                     86   D4 0001F          CLRL    (REC_ADDR)+              ;  1317
                                                     07   11 00021          BRB     2$                       ;  1306
                          86                         10   90 00023 1$:      MOVB    #16, (REC_ADDR)+         ;  1328
                          86          06             A5   90 00026          MOVB    6(BKT_ADDR), (REC_ADDR)+ ;  1331
                                      06             A5   96 0002A 2$:      INCB    6(BKT_ADDR)              ;  1337
                    15    1C    A7                   06   E0 0002D 3$:      BBS     #6, 28(IDX_DFN), 4$      ;  1353
                          86          20             A7   9B 00032          MOVZBW  32(IDX_DFN), (REC_ADDR)+ ;  1356
                                                     56   DD 00036          PUSHL   REC_ADDR                 ;  1359
                          50          00B4           CA   3C 00038          MOVZWL  180(IFAB), R0            ;
                                            60 B940  9F   9F 0003D          PUSHAB  @96(IRAB)[R0]            ;
                          7E          20             A7   9A 00041          MOVZBL  32(IDX_DFN), -(SP)       ;
```

```
                                37  11 00045                BRB     6$                                      ;
                          52    00B4   CA  3C 00047 4$:      MOVZWL  180(IFAB), KEYBUF                       ; 1379
                          52      60   A9  C0 0004C          ADDL2   96(IRAB), KEYBUF                        ;
                          50           52  D0 00050          MOVL    KEYBUF, R0                              ; 1380
                                0000G  30  00053              BSBW    RM$COMPRESS_KEY                         ;
                     50           56     14   AE  C1 00056   ADDL3   SIDR_SIZE, REC_ADDR, R0                 ; 1386
                          51        04   A5  3C 0005B        MOVZWL  4(BKT_ADDR), R1                         ;
                          51           55  C0 0005F          ADDL2   BKT_ADDR, R1                            ;
                          51           50  D1 00062          CMPL    R0, R1                                  ;
                                       0A  1E 00065          BGEQU   5$                                      ;
                          51        02   A0  9E 00067        MOVAB   2(R0), R1                               ; 1388
                          50           52  D0 0006B          MOVL    KEYBUF, R0                              ;
                                0000G  30  0006E              BSBW    RM$RECOMPR_KEY                          ;
                          50           62  9A 00071 5$:      MOVZBL  (KEYBUF), R0                            ; 1395
                          50           02  C0 00074          ADDL2   #2, R0                                  ;
                          86           50  B0 00077          MOVW    R0, (REC_ADDR)+                         ;
                                0045   0F  BB 0007A          PUSHR   #^M<R0,R2,R6>                           ; 1398
                                0C00G  30  0007D 6$:         BSBW    RM$MOVE                                 ;
                                       0C  C0 00081          ADDL2   #12, SP                                 ;
                          56           50  D0 00084          MOVL    R0, REC_ADDR                            ;
                          52           56  D0 00087          MOVL    REC_ADDR, FIRST_ARRAY_ADDR              ; 1405
                                FC87   30  0008A              BSBW    RM$ADD_TO_ARRAY                         ; 1406
                          03    00B7   CA  91 0008D          CMPB    183(IFAB), #3                           ; 1412
                                       05  1F 00092          BLSSU   7$                                      ;
                          04      43   A9  E8 00094          BLBS    67(IRAB), 7$                            ; 1414
                          62      80   8F  88 00098          BISB2   #128, (FIRST_ARRAY_ADDR)                ; 1416
                                0914   8F  BA 0009C 7$:      POPR    #^M<R2,R4,R8,R11>                       ; 1417
                                           05 000A0          RSB                                             ;

; Routine Size:  161 bytes,    Routine Base:  RM$RMS3 + 02EC
```

```
;1361        1418  1 %SBTTL  'RM$FOOLED_YUH'
;1362        1419  1 GLOBAL ROUTINE RM$FOOLED_YUH : RL$RABREG_7 =
;1363        1420  1
;1364        1421  1 !++
;1365        1422  1 !
;1366        1423  1 ! FUNCTIONAL DESCRIPTION:
;1367        1424  1 !
;1368        1425  1 ! routine to do all the probing, etc. to make sure the user hasn't
;1369        1426  1 ! been fooling around w/ rsz, rbf or the buffer describing his record
;1370        1427  1 ! between the time i started to position for insert and i go to insert
;1371        1428  1 !
;1372        1429  1 !
;1373        1430  1 ! CALLING SEQUENCE:
;1374        1431  1 !       bsbw rm$fooled_yuh
;1375        1432  1 !
;1376        1433  1 ! INPUT PARAMETERS:
;1377        1434  1 !       NONE
;1378        1435  1 !
;1379        1436  1 ! IMPLICIT INPUTS:
;1380        1437  1 !       irab -- mode, address of internal key buffer to check against
;1381        1438  1 !       IFAB -- kbufsz
;1382        1439  1 !       rab -- rsz, rbf
;1383        1440  1 !       idx_dfn -- minrecsz, keysz
;1384        1441  1 !
;1385        1442  1 ! OUTPUT PARAMETERS:
;1386        1443  1 !       NONE
;1387        1444  1 !
;1388        1445  1 ! IMPLICIT OUTPUTS:
;1389        1446  1 !       NONE
;1390        1447  1 !
;1391        1448  1 ! ROUTINE VALUE:
;1392        1449  1 !       rmssuc if the user has been considerate
;1393        1450  1 !       rsz or rbf if not
;1394        1451  1 !
;1395        1452  1 ! SIDE EFFECTS:
;1396        1453  1 !       ap is clobbered
;1397        1454  1 !
;1398        1455  1 !--
;1399        1456  1
;1400        1457  2     BEGIN
;1401        1458  2
;1402        1459  2     EXTERNAL REGISTER
;1403        1460  2         R_IFAB_STR,
;1404        1461  2         R_IRAB_STR,
;1405        1462  2         R_RAB_STR,
;1406        1463  2         R_IDX_DFN_STR;
;1407        1464  2
;1408        1465  2     LOCAL
;1409        1466  2         RBF_ADDR,
;1410        1467  2         STATUS;
;1411        1468  2
;1412        1469  2     BUILTIN
;1413        1470  2         AP;
;1414        1471  2
;1415        1472  2     ! do all the probing and comparing to make sure the user hasn't been
;1416        1473  2     ! fooling around w/ the record buffer
;1417        1474  2     !
```

```
;  1418          1475  2
;  1419          1476  2      IF .RAB[RAB$W_RSZ] LSSU .IDX_DFN[IDX$W_MINRECSZ]
;  1420          1477  2      THEN
;  1421          1478  2          RETURN RMSERR(RSZ);
;  1422          1479  2
;  1423          1480  2      RBF_ADDR    .RAB[RAB$L_RBF];
;  1424          1481  2      IF RMSNOREAD_LONG(.RAB[RAB$W_RSZ], .RBF_ADDR, .IRAB[IRB$B_MODE])
;  1425          1482  2      THEN
;  1426          1483  2          RETURN RMSERR(RBF);
;  1427          1484  2
;  1428          1485  2      AP = 2;
;  1429          1486  2
;  1430          1487  2      ! If keys do not match, return error
;  1431          1488  2      !
;  1432          1489  2
;  1433          1490  2      IF RMSCOMPARE_KEY(.RBF_ADDR,
;  1434          1491  2                        KEYBUF_ADDR(3),
;  1435          1492  2                        .IDX_DFN[IDX$B_KEYSZ])
;  1436          1493  2      THEN
;  1437          1494  2          RETURN RMSERR(RBF);
;  1438          1495  2
;  1439          1496  3      RETURN RMSSUC()
;  1440          1497  3
;  1441          1498  1      END;

                        0C   BB 00000 RMSFOOLED_YUH::
                                      PUSHR   #^M<R2,R3>                          ;  1419
          22    A7      22   A8  B1 00002      CMPW    34(RAB), 34(IDX_DFN)       ;  1476
                        07   1E  1E 00007      BGEQU   1$
                50    86A4   8F  3C 00009      MOVZWL  #34468, R0                 ;  1478
                        3B   11  0000E         BRB     4$
                52      28   A8  D0 00010 1$:  MOVL    40(RAB), RBF_ADDR          ;  1480
                7E      0A   A9  9A 00014      MOVZBL  10(IRAB), -(SP)            ;  1481
                        52   DD  00018         PUSHL   RBF_ADDR
                7E      22   A8  3C 0001A      MOVZWL  34(RAB), -(SP)
                          0000G 30  0001E       BSBW    RMSNOREAD_LONG
                5E            0C  C0 00021    ADDL2   #12, SP
                1A            50  E8 00024    BLBS    R0, 2$
                5C            02  D0 00027    MOVL    #2, AP                     ;  1485
                50    00B4    CA  3C 0002A    MOVZWL  180(IFAB), R0              ;  1491
                53      60 B940   3E  0002F   MOVAW   @96(IRAB)[R0], R3
                50      20    A7  9A 00034    MOVZBL  32(IDX_DFN), R0            ;  1490
                51            52  D0 00038    MOVL    RBF_ADDR, R1
                          0000G 30  0003B       BSBW    RMSCOMPARE_KEY
                07            50  E9 0003E    BLBC    R0, 3$
                50    8654    8F  3C 00041 2$:  MOVZWL  #34388, R0                ;  1494
                        03   11  00046         BRB     4$
                50            01  D0 00048 3$:  MOVL    #1, R0                     ;  1496
                              0C  BA 0004B 4$:  POPR    #^M<R2,R3>                 ;  1498
                                  05 0004D      RSB

; Routine Size:  78 bytes,    Routine Base:  RMSRMS3 + 038D
```

```
;1443    1499  1 %SBTTL  'RM$INS_REC'
;1444    1500  1 GLOBAL ROUTINE RM$INS_REC (BUCKET, REC_SZ) : RL$RABREG_67 =
;1445    1501  1
;1446    1502  1 !++
;1447    1503  1 !
;1448    1504  1 ! FUNCTIONAL DESCRIPTION:
;1449    1505  1 !
;1450    1506  1 ! Insert new record into bucket.  Check for ID availability if required,
;1451    1507  1 ! make room by spreading existing records apart, and call appropriate
;1452    1508  1 ! routine to build the record.
;1453    1509  1 !
;1454    1510  1 ! CALLING SEQUENCE:
;1455    1511  1 !
;1456    1512  1 !       RM$INS_REC()
;1457    1513  1 !
;1458    1514  1 ! INPUT PARAMETERS:
;1459    1515  1 !
;1460    1516  1 !       BKT_ADDR - address of bucket in which to insert record
;1461    1517  1 !
;1462    1518  1 !       REC_SZ - total size of record to be inserted
;1463    1519  1 !
;1464    1520  1 ! IMPLICIT INPUTS:
;1465    1521  1 !
;1466    1522  1 !       REC_ADDR - is where to build the record
;1467    1523  1 !
;1468    1524  1 !       IRAB - internal stream context
;1469    1525  1 !           [dups_seen] - this is a duplicate record
;1470    1526  1 !           [spl_idx] - left hand part only of index record
;1471    1527  1 !           [pos_ins] - offset to position of insert
;1472    1528  1 !
;1473    1529  1 !       IFAB - internal file context, used by called routines
;1474    1530  1 !
;1475    1531  1 !       IDX_DFN - index descriptor, used by called routines
;1476    1532  1 !
;1477    1533  1 ! OUTPUT PARAMETERS:
;1478    1534  1 !       NONE
;1479    1535  1 !
;1480    1536  1 ! IMPLICIT OUTPUTS:
;1481    1537  1 !       NONE
;1482    1538  1 !
;1483    1539  1 ! ROUTINE VALUE:
;1484    1540  1 !       success
;1485    1541  1 !       low bit clear if no id's available
;1486    1542  1 !
;1487    1543  1 !
;1488    1544  1 !--
;1489    1545  1
;1490    1546  2     BEGIN
;1491    1547  2
;1492    1548  2     EXTERNAL REGISTER
;1493    1549  2         COMMON_RAB_STR,
;1494    1550  2         R_REC_ADDR_STR,
;1495    1551  2         R_IDX_DFN_STR;
;1496    1552  2
;1497    1553  2     GLOBAL REGISTER
;1498    1554  2         R_BKT_ADDR_STR;
;1499    1555  2
```

```
;1500    1556  2     LOCAL
;1501    1557  2         TEMP    : BBLOCK [6];
;1502    1558  2
;1503    1559  2     MACRO
;1504    1560  2         OLD_OVHD        = TEMP[0,0,16,0] %,
;1505    1561  2         LENGTH          = TEMP[2,0,32,0] %,
;1506    1562  2         KEY_SZ          = REC_SZ<0,16> %,
;1507    1563  2         VBN_SZ          = REC_SZ<16,16>%;
;1508    1564  2
;1509    1565  2     BKT_ADDR = .BUCKET;
;1510    1566  2
;1511    1567  2     OLD_OVHD = 0;
;1512    1568  2
;1513    1569  2     IF .IFAB [IFB$B_PLG_VER] LSSU PLG$C_VER_3
;1514    1570  2     THEN
;1515    1571  2
;1516    1572  3         BEGIN
;1517    1573  3
;1518    1574  3         IF .BKT_ADDR [BKT$B_LEVEL] EQL 0
;1519    1575  3         THEN
;1520    1576  4             BEGIN
;1521    1577  4
;1522    1578  4             IF NOT .IRAB[IRB$V_DUPS_SEEN]
;1523    1579  4                     AND
;1524    1580  5                         (.BKT_ADDR [BKT$B_NXTRECID] EQL 0
;1525    1581  5                         OR
;1526    1582  5                         (.BKT_ADDR [BKT$B_NXTRECID] GTRU .BKT_ADDR [BKT$B_LSTRECID]))
;1527    1583  4             THEN
;1528    1584  4                 RETURN_ON_ERROR (RM$BKT_SORT(.BKT_ADDR));
;1529    1585  4
;1530    1586  4             END
;1531    1587  3         ELSE
;1532    1588  3             IF NOT .IRAB[IRB$V_SPL_IDX]
;1533    1589  3             THEN
;1534    1590  3                 OLD_OVHD = .REC_ADDR[IRC$V_PTRSZ] + IRC$C_IDXPTRBAS +
;1535    1591  3                            IRC$C_IDXOVHDSZ;
;1536    1592  2         END;
;1537    1593  2
;1538    1594  2     ! Figure out length of data to move and do it if non-zero - it will
;1539    1595  2     ! be zero when SIDR record is being inserted at the end of the bucket
;1540    1596  2     !
;1541    1597  2     !
;1542    1598  2     LENGTH = .BKT_ADDR[BKT$W_FREESPACE] - .IRAB[IRB$W_POS_INS] - .OLD_OVHD;
;1543    1599  2
;1544    1600  3     IF (.LENGTH GTR 0)
;1545    1601  2         AND
;1546    1602  3        (.KEY_SZ GTR 0)
;1547    1603  2     THEN
;1548    1604  3         BEGIN
;1549    1605  3
;1550    1606  3         GLOBAL REGISTER
;1551    1607  3             R_BDB;
;1552    1608  3
;1553    1609  3         RMSMOVE(.LENGTH, .REC_ADDR + .OLD_OVHD,
;1554    1610  3             .REC_ADDR + .KEY_SZ + .OLD_OVHD);
;1555    1611  2         END;
;1556    1612  2
```

```
;  1557          1613  2     ! Update the freespace pointer to reflect new size of data area
;  1558          1614  2     !
;  1559          1615  2     BKT_ADDR [BKT$W_FREESPACE] = .BKT_ADDR [BKT$W_FREESPACE] + .KEY_SZ;
;  1560          1616  2
;  1561          1617  3     IF  (.IFAB [IFB$B_PLG_VER] GEQU PLG$C_VER_3)
;  1562          1618  2          AND
;  1563          1619  3         (.BKT_ADDR[BKT$B_LEVEL] GTRU 0)
;  1564          1620  2     THEN
;  1565          1621  2
;  1566          1622  2         ! We must spread apart the VBN list for prologue 3 index records, also.
;  1567          1623  2         !
;  1568          1624  2         RM$SHFT_VBNS();
;  1569          1625  2
;  1570          1626  2     ! Insert new index record and return success.
;  1571          1627  2     !
;  1572          1628  2     IF .BKT_ADDR [BKT$B_LEVEL] NEQU 0
;  1573          1629  2     THEN
;  1574          1630  3         BEGIN
;  1575          1631  3         RM$BLD_IDX_REC (.REC_SZ);
;  1576          1632  3         RETURN 1
;  1577          1633  2         END;
;  1578          1634  2
;  1579          1635  2     ! If duplicates have been seen, then RMS merely has to add a new SIDR
;  1580          1636  2     ! array element to an existing array, otherwise, it must create a new
;  1581          1637  2     ! SIDR.
;  1582          1638  2     !
;  1583          1639  2     IF .IRAB[IRB$V_DUPS_SEEN]
;  1584          1640  2     THEN
;  1585          1641  2         RM$ADD_TO_ARRAY()
;  1586          1642  2     ELSE
;  1587          1643  2         RM$BLD_NEW_SIDR (.REC_SZ);
;  1588          1644  2
;  1589          1645  2     RETURN 1
;  1590          1646  1     END;


                                    30  BB 00000 RM$INS_REC::
                                                  PUSHR   #^M<R4,R5>                          ; 1500
                               5E                 08  C2 00002          SUBL2   #8, SP
                               55          14     AE  D0 00005          MOVL    BUCKET, BKT_ADDR    ; 1565
                                                  6E  B4 00009          CLRW    TEMP                ; 1567
                               03        00B7     CA  91 0000B          CMPB    183(IFAB), #3       ; 1569
                                                  30  1E 00010          BGEQU   3$
                                          0C     A5  95 00012          TSTB    12(BKT_ADDR)        ; 1574
                                                  1E  12 00015          BNEQ    2$
                                          44     A9  95 00017          TSTB    68(IRAB)            ; 1578
                                                  26  19 0001A          BLSS    3$
                                          06     A5  95 0001C          TSTB    6(BKT_ADDR)         ; 1580
                                                  07  13 0001F          BEQL    1$
                                    07    A5    06  A5  91 00021          CMPB    6(BKT_ADDR), 7(BKT_ADDR)    ; 1582
                                                  1A  1B 00026          BLEQU   3$
                                                  55  DD 00028 1$:      PUSHL   BKT_ADDR            ; 1584
                                                FD81  30 0002A          BSBW    RM$BKT_SORT
                               5E                 04  C0 0002D          ADDL2   #4, SP
```

```
                                 0F              50  E8  00030          BLBS    STATUS, 3$                                      ;
                                                 77  11  00033          BRB     10$                                             ;
                                 09          44  A9  E8  00035  2$:     BLBS    68(IRAB), 3$                                    ; 1588
                        50               66  02      00  EF  00039      EXTZV   #0, #2, (REC_ADDR), R0                          ; 1590
                                         6E  50      03  A1  0003E      ADDW3   #3, R0, TEMP                                    ;
                                 50          04  A5  3C  00042  3$:     MOVZWL  4(BKT_ADDR), R0                                 ; 1598
                                 51          48  A9  3C  00046          MOVZWL  72(IRAB), R1                                    ;
                                 50              51  C2  0004A          SUBL2   R1, R0                                          ;
                                 54              6E  3C  0004D          MOVZWL  TEMP, R4                                        ;
                                 50          02  AE  C3  00050          SUBL3   R4, R0, TEMP+2                                  ;
                                                 1E  15  00055          BLEQ    4$                                              ; 1600
                                             18  AE  B5  00057          TSTW    REC_SZ                                          ; 1602
                                                 19  13  0005A          BEQL    4$                                              ;
                                 50          18  AE  3C  0005C          MOVZWL  REC_SZ, R0                                      ; 1610
                                 50              56  C0  0C060          ADDL2   REC_ADR, R0                                     ;
                                 51              6E  3C  00063          MOVZWL  TEMP, R1                                        ;
                                                 6140  9F  00066        PUSHAB  (R1)[R0]                                        ;
                                                 6146  9F  00069        PUSHAB  (R1)[REC_ADDR]                                  ; 1609
                                             0A  AE  DD  0006C          PUSHL   TEMP+2                                          ;
                                                 0000G 30  0006F        BSBW    RM$MOVE                                         ;
                                 5E              0C  C0  00072          ADDL2   #12, SP                                         ;
                             04  A5          18  AE  A0  00075  4$:     ADDW2   REC_SZ, 4(BKT_ADDR)                             ; 1615
                                 03          00B7  CA  91  0007A        CMPB    183(IFAB), #3                                   ; 1617
                                                 08  1F  0007F          BLSSU   5$                                              ;
                                             0C  A5  95  00081          TSTB    12(BKT_ADDR)                                    ; 1619
                                                 03  13  00084          BEQL    5$                                              ;
                                                 0000V 30  00086        BSBW    RM$SHFT_VBNS                                    ; 1624
                                             0C  A5  95  00089  5$:     TSTB    12(BKT_ADDR)                                    ; 1628
                                                 08  13  0008C          BEQL    6$                                              ;
                                             18  AE  DD  0008E          PUSHL   REC_SZ                                          ; 1631
                                                 FDAC  30  00091        BSBW    RM$BLD_IDX_REC                                  ;
                                                 10  11  00094          BRB     8$                                              ;
                                             44  A9  95  00096  6$:     TSTB    68(IRAB)                                        ; 1639
                                                 05  18  00099          BGEQ    7$                                              ;
                                                 FB87  30  0009B        BSBW    RM$ADD_TO_ARRAY                                 ; 1641
                                                 09  11  0009E          BRB     9$                                              ;
                                             18  AE  DD  000A0  7$:     PUSHL   REC_SZ                                          ; 1643
                                                 FE6B  30  000A3        BSBW    RM$BLD_NEW_SIDR                                 ;
                             5E                  04  C0  000A6  8$:     ADDL2   #4, SP                                          ;
                             50                  01  D0  000A9  9$:     MOVL    #1, R0                                          ; 1645
                             5E                  08  C0  000AC  10$:    ADDL2   #8, SP                                          ; 1646
                                                 30  BA  000AF          POPR    #^M<R4,R5>                                      ;
                                                     05  000B1          RSB                                                     ;

; Routine Size:  178 bytes,    Routine Base:  RM$RMS3 + 03DB
```

```
;  1592     1647  1 %SBTTL  'RM$MOVE_IN_VBN'
;  1593     1648  1 ROUTINE RM$MOVE_IN_VBN (VBN) : RL$MOVE_IN_VBN NOVALUE =
;  1594     1649  1
;  1595     1650  1 !++
;  1596     1651  1 !
;  1597     1652  1 ! FUNCTIONAL DESCRIPTION:
;  1598     1653  1 !
;  1599     1654  1 ! set pointer size field and move in VBN, updating REC_ADDR
;  1600     1655  1 !
;  1601     1656  1 ! CALLING SEQUENCE:
;  1602     1657  1 !       RM$MOVE_IN_VBN(VBN)
;  1603     1658  1 !
;  1604     1659  1 ! INPUT PARAMETERS:
;  1605     1660  1 !       VBN - VBN to move in
;  1606     1661  1 !
;  1607     1662  1 ! IMPLICIT INPUTS:
;  1608     1663  1 !       REC_ADDR - address to move value to
;  1609     1664  1 !
;  1610     1665  1 ! OUTPUT PARAMETERS:
;  1611     1666  1 !       NONE
;  1612     1667  1 !
;  1613     1668  1 ! IMPLICIT OUTPUTS:
;  1614     1669  1 !       REC_ADDR - points one byte beyond where value placed
;  1615     1670  1 !
;  1616     1671  1 ! ROUTINE VALUE:
;  1617     1672  1 !       NONE
;  1618     1673  1 !
;  1619     1674  1 ! SIDE EFFECTS:
;  1620     1675  1 !       NONE
;  1621     1676  1 !
;  1622     1677  1 !--
;  1623     1678  1
;  1624     1679  2     BEGIN
;  1625     1680  2
;  1626     1681  2     EXTERNAL REGISTER
;  1627     1682  2         R_REC_ADDR;
;  1628     1683  2
;  1629     1684  2     GLOBAL REGISTER
;  1630     1685  2         COMMON_RABREG,
;  1631     1686  2         R_BDB,
;  1632     1687  2         R_IDX_DFN;
;  1633     1688  2
;  1634     1689  3     BEGIN
;  1635     1690  3
;  1636     1691  3     REGISTER
;  1637     1692  3         SIZE = 1        : BYTE;
;  1638     1693  3
;  1639     1694  3     ! get vbn size and set pointer field
;  1640     1695  3     !
;  1641     1696  3     SIZE = RM$VBN_SIZE(.VBN);
;  1642     1697  3     (.REC_ADDR)<0, 8> = .SIZE - 2;
;  1643     1698  3     REC_ADDR = .REC_ADDR + 1;
;  1644     1699  3
;  1645     1700  3     ! stick in bucket pointer
;  1646     1701  3     !
;  1647     1702  3     (.REC_ADDR)<0, .SIZE*8> = .VBN;
;  1648     1703  3     REC_ADDR = .REC_ADDR + .SIZE;
```

```
;  1649          1704  2     END;
;  1650          1705  1     END;


                                0F94    8F   BB 00000 RM$MOVE_IN_VBN:
                                                      PUSHR   #^M<R2,R4,R7,R8,R9,R10,R11>     ;  1696
                                     0000V 30 00004           BSBW    RM$VBN_SIZE             ;
                                       5E    04 C0 00007      ADDL2   #4, SP                  ;
                                       51    50 90 0000A      MOVB    R0, SIZE
                                       51    02 83 0000D      SUBB3   #2, SIZE, (REC_ADDR)+       1697
                                       50    51 9A 00011      MOVZBL  SIZE, R0                ;  1702
                                       50    08 C4 00014      MULL2   #8, R0                  ;
                                       50    52 F0 00017      INSV    VBN, #0, R0, (REC_ADDR)
                                       50    51 9A 0001C      MOVZBL  SIZE, R0                ;  1703
                                       56    50 C0 0001F      ADDL2   R0, REC_ADDR            ;
                                0F90    8F   BA 00022          POPR    #^M<R4,R7,R8,R9,R10,R11>;  1705
                                             05 00026          RSB                             ;

; Routine Size:  39 bytes,    Routine Base:  RM$RMS3 + 048D
```

```
;  1652          1706  1 %SBTTL  'RM$NEW_VBN_BYTES'
;  1653          1707  1 ROUTINE RM$NEW_VBN_BYTES (VBN_SIZE) : RL$NEW_VBN_BYTES =
;  1654          1708  1 !++
;  1655          1709  1 !
;  1656          1710  1 ! RM$NEW_VBN_BYTES
;  1657          1711  1 !
;  1658          1712  1 !    This routine returns the number of additional bytes which will
;  1659          1713  1 !    be needed if all the VBNs grow to VBN_SIZE
;  1660          1714  1 !
;  1661          1715  1 ! CALLING SEQUENCE:
;  1662          1716  1 !
;  1663          1717  1 !       RM$NEW_VBN_BYTES (VBN_SIZE)
;  1664          1718  1 !
;  1665          1719  1 ! INPUT PARAMETERS:
;  1666          1720  1 !
;  1667          1721  1 !       VBN_SIZE - potential number of bytes per VBN
;  1668          1722  1 !
;  1669          1723  1 ! IMPLICIT INPUTS:
;  1670          1724  1 !
;  1671          1725  1 !       BKT_ADDR -  address of current bucket
;  1672          1726  1 !       IDX_DFN  -  index descriptor structure
;  1673          1727  1 !
;  1674          1728  1 ! OUTPUT PARAMETERS:
;  1675          1729  1 !
;  1676          1730  1 !       NONE
;  1677          1731  1 !
;  1678          1732  1 ! IMPLICIT OUTPUTS:
;  1679          1733  1 !       NONE
;  1680          1734  1 !
;  1681          1735  1 ! ROUTINE VALUE:
;  1682          1736  1 !
;  1683          1737  1 !       R0 - number of new bytes which will be needed if all the
;  1684          1738  1 !            VBNs in the bucket grow to be VBN_SIZE long
;  1685          1739  1 !
;  1686          1740  1 !--
;  1687          1741  1
;  1688          1742  2     BEGIN
;  1689          1743  2
;  1690          1744  2     MACRO
;  1691          1745  2         FREESPC = 0,0,16,0 %;
;  1692          1746  2
;  1693          1747  2     GLOBAL REGISTER
;  1694          1748  2         R_BKT_ADDR_STR,
;  1695          1749  2         R_IDX_DFN_STR;
;  1696          1750  2
;  1697          1751  2     LOCAL
;  1698          1752  2         CUR_VBN_SZ,
;  1699          1753  2         FIRST_VBN       : REF BBLOCK,
;  1700          1754  2         NUM_VBNS;
;  1701          1755  2
;  1702          1756  2     CUR_VBN_SZ = .BKT_ADDR [BKT$V_PTR_SZ] + 2;
;  1703          1757  2
;  1704          1758  2     ! If VBNs didn't grow return
;  1705          1759  2     !
;  1706          1760  2     IF .VBN_SIZE LEQU .CUR_VBN_SZ
;  1707          1761  2     THEN
;  1708          1762  2         RETURN 0;
```

```
;  1709          1763  2
;  1710          1764  2      ! Number of new bytes = (new_size - old_size) * num_vbns
;  1711          1765  2      !
;  1712          1766  2      FIRST_VBN = .BKT_ADDR + (.IDX_DFN [IDX$B_IDXBKTSZ] * 512) - BKT$C_ENDOVHD;
;  1713          1767  2      NUM_VBNS = (.FIRST_VBN - (.BKT_ADDR + .FIRST_VBN [FREESPC]))/ (.BKT_ADDR [BKT$V_PTR_SZ] + 2);
;  1714          1768  2      RETURN (.VBN_SIZE = .CUR_VBN_SZ) * .NUM_VBNS
;  1715          1769  2
;  1716          1770  1      END;
;  INFO#250                L1:1756
;  Referenced REGISTER symbol BKT_ADDR is probably not initialized
;  INFO#250                L1:1768
;  Referenced REGISTER symbol IDX_DFN is probably not initialized



                                    00A4    8F   BB 00000 RM$NEW_VBN_BYTES:
                                                           PUSHR   #^M<R2,R5,R7>                       ; 1707
          51        0D    A5              02             03   EF 00004          EXTZV   #3, #2, 13(BKT_ADDR), R1    ; 1756
                                  51        02             C0 0000A          ADDL2   #2, R1              ;
                                  52        51             D0 0000D          MOVL    R1, CUR_VBN_SZ      ;
                                  52        10    AE       D1 00010          CMPL    VBN_SIZE, CUR_VBN_SZ        ; 1760
                                            24             1B 00014          BLEQU   1$                  ;
                                  57        16    A7       9A 00016          MOVZBL  22(IDX_DFN), R7             ; 1766
                        57        57              09       78 0001A          ASHL    #9, R7, R7          ;
                                  50        FC A745        9E 0001E          MOVAB   -4(R7)[BKT_ADDR], FIRST_VBN ;
                                  57              60       3C 00023          MOVZWL  (FIRST_VBN), R7             ; 1767
                                  55              57       C0 00026          ADDL2   R7, BKT_ADDR        ;
                                  50              55       C2 00029          SUBL2   R5, R0              ;
                        55        50              51       C7 0002C          DIVL3   R1, R0, NUM_VBNS    ;
                        50              10    AE       52   C3 00030          SUBL3   CUR_VBN_SZ, VBN_SIZE, R0        ; 1768
                                  50              55       C4 00035          MULL2   NUM_VBNS, R0        ;
                                                  02       11 00038          BRB     2$                  ;
                                                  50       D4 0003A 1$:      CLRL    R0                  ; 1770
                                    00A4    8F    BA 0003C 2$:      POPR    #^M<R2,R5,R7>               ;
                                                  05 00040          RSB                                 ;

; Routine Size:  65 bytes,    Routine Base:  RM$RMS3 + 04B4
```

```
;  1718     1771  1 %SBTTL  'RM$RECORD_SIZE'
;  1719     1772  1 GLOBAL ROUTINE RM$RECORD_SIZE : RL$RABREG_567 =
;  1720     1773  1
;  1721     1774  1 !++
;  1722     1775  1 !
;  1723     1776  1 ! FUNCTIONAL DESCRIPTION:
;  1724     1777  1 !
;  1725     1778  1 ! Calculate record size of SIDR or index records
;  1726     1779  1 !
;  1727     1780  1 ! CALLING SEQUENCE:
;  1728     1781  1 !
;  1729     1782  1 !       RM$RECORD_SIZE()
;  1730     1783  1 !
;  1731     1784  1 ! INPUT PARAMETERS:
;  1732     1785  1 !       NONE
;  1733     1786  1 !
;  1734     1787  1 ! IMPLICIT INPUTS:
;  1735     1788  1 !       IFAB            - for RM$RECORD_VBN
;  1736     1789  1 !       IRAB            - pointer to internal RAB
;  1737     1790  1 !               [ STOPLEVEL ] - current level in tree
;  1738     1791  1 !               [ DUPS_SEEN ] - flag set if duplicates seen
;  1739     1792  1 !               [ CONT_BKT ] - this is continuation bucket
;  1740     1793  1 !               [ BIG_SPLIT ] - flag set if more than two bucket split
;  1741     1794  1 !               [ VBN_LEFT ]  - VBN of leftmost bucket in split
;  1742     1795  1 !               [ VBN_RIGHT ] - VBN of rightmost bucket in split
;  1743     1796  1 !               [ VBN_MID ] - VBN of middle bucket in 3-4 bucket split
;  1744     1797  1 !               [ REC_COUNT ] - which record this is in the bucket
;  1745     1798  1 !               [ POS_INS ]  - position of insert of this record
;  1746     1799  1 !               [ KEYBUF ] - pointer to the 5 contigious keybuffers
;  1747     1800  1 !               [ PUTUP_VBN ] - VBN of new primary data record (for SIDR ptr)
;  1748     1801  1 !               [ PUTUP_ID ] - ID of new primary data record (for SIDR ptr)
;  1749     1802  1 !       IDX_DFN         - pointer to index descriptor structure
;  1750     1803  1 !               [ DUPKEYS ] - duplicate keys are allowed
;  1751     1804  1 !               [ KEYSZ ] - size of key
;  1752     1805  1 !       REC_ADDR - position of insert (used on index levels only)
;  1753     1806  1 !       BKT_ADDR - pointer to current bucket
;  1754     1807  1 !
;  1755     1808  1 ! OUTPUT PARAMETERS:
;  1756     1809  1 !       NONE
;  1757     1810  1 !
;  1758     1811  1 ! IMPLICIT OUTPUTS:
;  1759     1812  1 !       NONE
;  1760     1813  1 !
;  1761     1814  1 ! ROUTINE VALUE:
;  1762     1815  1 !       size of record required
;  1763     1816  1 !
;  1764     1817  1 !       PROLOG 1 & 2:   Size is returned as a single quantity
;  1765     1818  1 !
;  1766     1819  1 !       PROLUG 3 Index: Size is returned as two contigious words.
;  1767     1820  1 !                       High order word contains number of VBN bytes
;  1768     1821  1 !                       Low order word contains number of key bytes
;  1769     1822  1 !
;  1770     1823  1 !       PROLOG 3 SIDR:  Size is returned as a single quantity
;  1771     1824  1 !
;  1772     1825  1 ! SIDE EFFECTS:
;  1773     1826  1 !
;  1774     1827  1 !       sets AP = 1 when checking down pointer on empty bucket cases
```

```
;  1775          1828  1 !       (index level).
;  1776          1829  1 !       Keybuffer 5 will be used if index/key compression is enabled.
;  1777          1830  1 !       Keybuffer 4 will be used if index compression is enabled and a big
;  1778          1831  1 !       split occurred.
;  1779          1832  1 !
;  1780          1833  1 !--
;  1781          1834  1
;  1782           835  2     BEGIN
;  1783          1836  2
;  1784          1837  2     BUILTIN
;  1785          1838  2         AP;
;  1786          1839  2
;  1787          1840  2     EXTERNAL REGISTER
;  1788          1841  2         COMMON_RAB_STR,
;  1789          1842  2         R_BKT_ADDR_STR,
;  1790          1843  2         R_REC_ADDR_STR,
;  1791          1844  2         R_IDX_DFN_STR;
;  1792          1845  2
;  1793          1846  2     GLOBAL REGISTER
;  1794          1847  2         R_BDB;
;  1795          1848  2
;  1796          1849  2     LOCAL
;  1797          1850  2         SIZE;
;  1798          1851  2
;  1799          1852  2     ! RMS is to determine the size of a new SIDR or SIDR array element.
;  1800          1853  2     !
;  1801          1854  2     IF .IRAB[IRB$B_STOPLEV[L] EQL 0
;  1802          1855  2     THEN
;  1803          1856  3         BEGIN
;  1804          1857  3
;  1805          1858  3         ! Calculate the size of the record pointer part of record (which will
;  1806          1859  3         ! always be present).
;  1807          1860  3         !
;  1808          1861  3         SIZE = RM$VBN_SIZE(.IRAB[IRB$L_PUTUP_VBN]);
;  1809          1862  3
;  1810          1863  3         IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
;  1811          1864  3         THEN
;  1812          1865  3             SIZE = .SIZE + IRC$C_DATOVHDSZ
;  1813          1866  3         ELSE
;  1814          1867  3             SIZE = .SIZE + IRC$C_DATOVHSZ3;
;  1815          1868  3
;  1816          1869  3
;  1817          1870  3         ! If a new SIDR is required, then add in the size of the key, and the
;  1818          1871  3         ! overhead associated with a SIDR.
;  1819          1872  3         !
;  1820          1873  3         IF NOT .IRAB[IRB$V_DUPS_SEEN]
;  1821          1874  3         THEN
;  1822          1875  3
;  1823          1876  3             ! Prologue 1 and 2 SIDR.
;  1824          1877  3             !
;  1825          1878  3             IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
;  1826          1879  3             THEN
;  1827          1880  4                 BEGIN
;  1828          1881  4
;  1829          1882  4                 ! If duplicates are allowed, then the overhead for the SIDR
;  1830          1883  4                 ! includes a four byte duplicate count field which will never
;  1831          1884  4                 ! be used but is there for compatibility. This field is
```

```
;  1832          1885  4                ! unnecessary, if the new SIDR is to go into a continuation
;  1833          1886  4                ! bucket.
;  1834          1887  4                !
;  1835          1888  4                IF .IDX_DFN[IDX$V_DUPKEYS]
;  1836          1889  4                    AND
;  1837          1890  4                   NOT .IRAB[IRB$V_CONT_BKT]
;  1838          1891  4                THEN
;  1839          1892  4                    SIZE = .SIZE + IRC$C_DATOVHDSZ + IRC$C_DATSZFLD
;  1840          1893  4                                                   + IRC$C_DCNTSZFLD
;  1841          1894  4                ELSE
;  1842          1895  4                    SIZE = .SIZE + IRC$C_DATOVHDSZ + IRC$C_DATSZFLD;
;  1843          1896  4
;  1844          1897  4                SIZE = .SIZE + .IDX_DFN[IDX$B_KEYSZ];
;  1845          1898  4                END
;  1846          1899  4
;  1847          1900  4            ! Prologue 3 SIDR.
;  1848          1901  4            !
;  1849          1902  3            ELSE
;  1850          1903  4                BEGIN
;  1851          1904  4                SIZE = .SIZE + IRC$C_SDROVHSZ3 + RM$V3KEY_SZ (KEYBUF_ADDR(2));
;  1852          1905  4
;  1853          1906  4                IF .IDX_DFN[IDX$V_KEY_COMPR]
;  1854          1907  4                THEN
;  1855          1908  4                    SIZE = .SIZE + IRC$C_KEYCMPOVH;
;  1856          1909  3                END;
;  1857          1910  3        END                                     ! of SIDR data level
```

```
;  1859          1911  3
;  1860          1912  2      ELSE
;  1861          1913  2
;  1862          1914  2          IF .IFAB [IFB$B_PLG_VER] LSSU PLG$C_VER_3
;  1863          1915  2          THEN
;  1864          1916  2
;  1865          1917  2              ! This is a Prolog 1 or 2 index bucket. The size to be returned is
;  1866          1918  2              ! given below for each of the possible split cases.
;  1867          1919  2              !
;  1868          1920  2              !  1) two-bucket split no empty buckets:
;  1869          1921  2              !
;  1870          1922  2              !     size of new key + size of VBN_RIGHT
;  1871          1923  2              !
;  1872          1924  2              !  2) two-bucket split empty bucket:
;  1873          1925  2              !
;  1874          1926  2              !     size of VBN_MID - size of old VBN in record
;  1875          1927  2              !     (only update VBN pointer)
;  1876          1928  2              !
;  1877          1929  2              !  3) multibucket split no empty buckets:
;  1878          1930  2              !
;  1879          1931  2              !     size of two new keys + size of first new VBN (VBN_MID)
;  1880          1932  2              !                          + size of second new VBN (VBN_RIGHT)
;  1881          1933  2              !
;  1882          1934  2              !  4) multibucket split with empty buckets:
;  1883          1935  2              !
;  1884          1936  2              !     size of two new keys + size of VBN_RIGHT
;  1885          1937  2              !                          + size of VBN_MID
;  1886          1938  2              !                          - size of old VBN in record
;  1887          1939  2              !     (update one VBN pointer)
;  1888          1940  2              !
;  1889          1941  2              ! Two-pass Split Cases Without Empty Buckets.
;  1890          1942  2              !
;  1891          1943  2              !  5) first pass of two-pass two-bucket split case:
;  1892          1944  2              !
;  1893          1945  2              !     size of new key + size of old VBN in record
;  1894          1946  2              !
;  1895          1947  2              !  6) second pass of two-pass two-bucket split case:
;  1896          1948  2              !
;  1897          1949  2              !     size of VBN_RIGHT - size of old VBN in record
;  1898          1950  2              !     (only update VBN pointer)
;  1899          1951  2              !
;  1900          1952  2              !  7) first pass of two-pass multibucket split case -
;  1901          1953  2              !     one of the new keys in each bucket:
;  1902          1954  2              !
;  1903          1955  2              !     size of first new key + size of old VBN in record
;  1904          1956  2              !
;  1905          1957  2              !  8) second pass of two-pass multibucket split case -
;  1906          1958  2              !     one of the new keys in each bucket:
;  1907          1959  2              !
;  1908          1960  2              !     size of second new key + size of new VBN (VBN_MID)
;  1909          1961  2              !                            + size of VBN_RIGHT
;  1910          1962  2              !                            - size of old VBN in record
;  1911          1963  2              !     (update one VBN pointer)
;  1912          1964  2              !
;  1913          1965  2              !  9) first pass of two-pass multibucket split case -
;  1914          1966  2              !     both of the new keys in the same bucket:
;  1915          1967  2              !
```

```
;  1916          1968  2              !       size of the two new keys + size of new VBN (VBN_MID)
;  1917          1969  2              !                                + size of old VBN in record
;  1918          1970  2              !
;  1919          1971  2              ! 10) second pass of two-pass multibucket split case -
;  1920          1972  2              !       both of the new keys in the same bucket:
;  1921          1973  2              !
;  1922          1974  2              !       size of VBN_RIGHT - size of old VBN in record
;  1923          1975  2              !       (only update VBN pointer)
;  1924          1976  2              !
;  1925          1977  2              ! Two-pass Bucket Split Cases with Empty Buckets
;  1926          1978  2              !
;  1927          1979  2              ! 11) second pass of two-pass two-bucket split case:
;  1928          1980  2              !
;  1929          1981  2              !       size of VBN_MID - size of old VBN in record
;  1930          1982  2              !       (only update VBN pointer)
;  1931          1983  2              !
;  1932          1984  2              ! 11) first pass of two-pass multi-bucket split case:
;  1933          1985  2              !
;  1934          1986  2              !       size of new key + size of VBN_MID
;  1935          1987  2              !                       - size of old VBN in record
;  1936          1988  2              !       (only update VBN in pointer)
;  1937          1989  2              !
;  1938          1990  2              ! 13) second pass of two-pass multi-bucket split case:
;  1939          1991  2              !
;  1940          1992  2              !       size of VBN_RIGHT - size of old VBN in record
;  1941          1993  2              !       (update VBN pointer)
;  1942          1994  2              !
;  1943          1995  2              ! sizes are always computed so as to include any control bytes
;  1944          1996  2              ! required
;  1945          1997  2              !
;  1946          1998  3              BEGIN
;  1947          1999  3              SIZE = 0;
;  1948          2000  3
;  1949          2001  3              ! Handle all empty bucket split cases
;  1950          2002  3              !
;  1951          2003  3              IF .IRAB[IRB$V_EMPTY_BKT]
;  1952          2004  3              THEN
;  1953          2005  4                  BEGIN
;  1954          2006  4                  AP = 1;                         ! set for index level
;  1955          2007  4
;  1956          2008  4                  ! In all empty bucket split cases the existing down pointer
;  1957          2009  4                  ! must match VBN_LEFT before we modify it. If it doesn't
;  1958          2010  4                  ! match we can't change it or we may be causing crossed
;  1959          2011  4                  ! pointers down to the level below.
;  1960          2012  4                  !
;  1961          2013  4                  IF .IRAB[IRB$L_VBN_LEFT] EQLA RM$RECORD_VBN()
;  1962          2014  4                  THEN
;  1963          2015  4
;  1964          2016  4                      ! If this is the first pass of the two-pass multi-bucket
;  1965          2017  4                      ! with empty buck' split case then if we are going to
;  1966          2018  4                      ! swing the poi     the size is IRB$L_VBN_MID.
;  1967          2019  4                      !
;  1968          2020  4                      IF .IRAB[IRB$V_SPL_IDX]
;  1969          2021  4                      THEN
;  1970          2022  4                          SIZE = RM$VBN_SIZE (.IRAB[IRB$L_VBN_MID])
;  1971          2023  4
;  1972          2024  4                      ! Otherwise we are just going to change the VBN pointer in
```

```
;1973   2025  4                ! place and the size is the difference between the current
;1974   2026  4                ! size, and the size of the VBN that wuill replace it
;1975   2027  4                ! (found in IRB$L_VBN_MID).
;1976   2028  4                !
;1977   2029  4                ELSE
;1978   2030  4                    SIZE = RM$VBN_SIZE(.IRAB[IRB$L_VBN_MID]) -
;1979   2031  4                           .REC_ADDR[IRC$V_PTRSZ] - 2;
;1980   2032  4
;1981   2033  4            IF NOT .IRAB[IRB$V_BIG_SPLIT]
;1982   2034  4            THEN
;1983   2035  4                RETURN .SIZE;
;1984   2036  4
;1985   2037  4            ! If this is a sizing for the second pass of a two-pass
;1986   2038  4            ! multi-bucket empty bucket split case, then the size of
;1987   2039  4            ! the index record to be inserted is only the difference
;1988   2040  4            ! between the size of the current record's VBN and the size
;1989   2041  4            ! of the VBN that is to replace it.
;1990   2042  4            !
;1991   2043  5            IF (.IRAB[IRB$L_VBN_LEFT] EQLA 0)
;1992   2044  4            THEN
;1993   2045  4                SIZE = RM$VBN_SIZE(.IRAB[IRB$L_VBN_RIGHT])
;1994   2046  4                           - .REC_ADDR[IRC$V_PTRSZ] - 2
;1995   2047  4
;1996   2048  4            ! For all other multi-bucket empty bucket split cases, the
;1997   2049  4            ! size of the index record(s) to be inserted includes the
;1998   2050  4            ! key size, and in the case of the one-pass multi-bucket
;1999   2051  4            ! empty bucket split case includes the size of the VBN in
;2000   2052  4            ! IRB$L_VBN_RIGHT.
;2001   2053  4            !
;2002   2054  4            ELSE
;2003   2055  5                BEGIN
;2004   2056  5                SIZE = .SIZE + .IDX_DFN[IDX$B_KEYSZ] + 1;
;2005   2057  5
;2006   2058  5                IF NOT .IRAB[IRB$V_SPL_IDX]
;2007   2059  5                THEN
;2008   2060  5                    SIZE = .SIZE + RM$VBN_SIZE(.IRAB[IRB$L_VBN_RIGHT]);
;2009   2061  4                END;
;2010   2062  4
;2011   2063  4                RETURN .SIZE;
;2012   2064  3                END;
;2013   2065  3
;2014   2066  3
;2015   2067  3            ! Determine size of index records to be added for all bucket split
;2016   2068  3            ! cases not involving empty buckets.
;2017   2069  3            !
;2018   2070  3            ! If this is not the first pass of a two-pass bucket split case,
;2019   2071  3            ! compute the difference between the current record VBN pointer
;2020   2072  3            ! size and the size of the VBN to swing (VBN_RIGHT). This will
;2021   2073  3            ! facilitate computation of the record size during the second
;2022   2074  3            ! pass of all two-pass bucket split cases when a VBN pointer must
;2023   2075  3            ! be updated (in fact becomes the size for two of the two-pass
;2024   2076  3            ! bucket split cases), and it becomes part of the compuation
;2025   2077  3            ! of the VBN pointer size required for VBN_RIGHT for all one-pass
;2026   2078  3            ! bucket split cases.
;2027   2079  3            !
;2028   2080  3            IF NOT .IRAB[IRB$V_SPL_IDX]
;2029   2081  3            THEN
```

```
;  2030          2082  3              SIZE = RM$VBN_SIZE(.IRAB[IRB$L_VBN_RIGHT])
;  2031          2083  3                   - .REC_ADDR[IRC$V_PTRSZ] - 2;
;  2032          2084  3
;  2033          2085  3          ! The size of the middle key key and its corresponding VBN must be
;  2034          2086  3          ! included in the computation of record size if this is a one-pass
;  2035          2087  3          ! multibucket case, the first pass of the two-pass multibucket
;  2036          2088  3          ! split case where both keys go in the same (old/left) index
;  2037          2089  3          ! bucket, or the second pass of the two-pass multibucket split
;  2038          2090  3          ! case where one of the new keys goes in each of the old and new
;  2039          2091  3          ! index buckets and we are currently updating the new index
;  2040          2092  3          ! bucket with the second of the two keys and its corresponding VBN.
;  2041          2093  3          !
;  2042          2094  4          IF (.IRAB[IRB$V_BIG_SPLIT] AND NOT .IRAB[IRB$V_SPL_IDX] AND
;  2043          2095  4                   .IRAB[IRB$L_VBN_MID] NEQU 0)
;  2044          2096  3              OR
;  2045          2097  4             (.IRAB[IRB$V_SPL_IDX]
;  2046          2098  4                   AND .IRAB[IRB$V_BIG_SPLIT]
;  2047          2099  4                   AND (.IRAB[IRB$W_POS_INS] EQLU BKT$C_OVERHDSZ))
;  2048          2100  3          THEN
;  2049          2101  3              SIZE = .SIZE + RM$VBN_SIZE(.IRAB[IRB$L_VBN_MID]) +
;  2050          2102  3                     .IDX_DFN[IDX$B_KEYSZ] + 1;
;  2051          2103  3
;  2052          2104  3          ! Finally, the size of the record must include the size of the
;  2053          2105  3          ! first key on all one-pass bucket split cases not involving
;  2054          2106  3          ! empty buckets, and on all two-pass bucket split cases during
;  2055          2107  3          ! the first pass when the contents of the old index buckets
;  2056          2108  3          ! are being updated.
;  2057          2109  3          !
;  2058          2110  3          ! Note that the current record size is also added in. If this is a
;  2059          2111  3          ! one-pass bucket split cases this quantity plus the difference
;  2060          2112  3          ! between the VBN pointer size in the old record and the VBN
;  2061          2113  3          ! pointer size required for VBN_RIGHT (already included within
;  2062          2114  3          ! SIZE) yield the size of VBN_RIGHT. If this is one of the two-pass
;  2063          2115  3          ! bucket split cases, it is at this point that we include in the
;  2064          2116  3          ! size determination the size of the old record's VBN pointer.
;  2065          2117  3          !
;  2066          2118  3          IF .IRAB[IRB$L_VBN_LEFT] NEQ 0
;  2067          2119  3          THEN
;  2068          2120  3              SIZE = .SIZE + .REC_ADDR[IRC$V_PTRSZ] + 2 +
;  2069          2121  3                     .IDX_DFN[IDX$B_KEYSZ] + 1;
;  2070          2122  3
;  2071          2123  3          END
```

```
;  2073          2124  3
;  2074          2125  2          ELSE
;  2075          2126  2
;  2076          2127  2              !This is a Prolog 3 index bucket. The size to be returned is
;  2077          2128  2              !given below for each of the possible split cases. Note that it
;  2078          2129  2              !will always be possible for the VBN pointer size to change in
;  2079          2130  2              !a bucket. This requires that the additional bytes needed to
;  2080          2131  2              !increase the VBN pointer size of every currently existing index
;  2081          2132  2              !record in the current bucket to the new value be included in the
;  2082          2133  2              !size determination.
;  2083          2134  2              !
;  2084          2135  2              ! 1) two-bucket split no empty buckets:
;  2085          2136  2              !
;  2086          2137  2              !    size of new key + VBN pointer size of bucket
;  2087          2138  2              !                    + number of new VBN bytes
;  2088          2139  2              !
;  2089          2140  2              ! 2) two-bucket split empty bucket:
;  2090          2141  2              !
;  2091          2142  2              !    number of new VBN bytes
;  2092          2143  2              !    (only update VBN pointer)
;  2093          2144  2              !
;  2094          2145  2              ! 3) multibucket split no empty buckets:
;  2095          2146  2              !
;  2096          2147  2              !    size of two new keys + 2 * VBN pointer size of bucket
;  2097          2148  2              !                         + number of new VBN bytes
;  2098          2149  2              !
;  2099          2150  2              ! 4) multibucket split with empty buckets:
;  2100          2151  2              !
;  2101          2152  2              !    size of two new keys + VBN pointer size of bucket
;  2102          2153  2              !                         + number of new VBN bytes
;  2103          2154  2              !    (update one VBN pointer)
;  2104          2155  2              !
;  2105          2156  2              !Two-pass Bucket Split Cases Without Empty Buckets
;  2106          2157  2              !
;  2107          2158  2              ! 5) first pass of two-pass two-bucket split case:
;  2108          2159  2              !
;  2109          2160  2              !    size of new key + VBN pointer size of bucket
;  2110          2161  2              !                    number of new VBN bytes
;  2111          2162  2              !
;  2112          2163  2              ! 6) second pass of two-pass two-bucket split case:
;  2113          2164  2              !
;  2114          2165  2              !    number of new VBN bytes
;  2115          2166  2              !    (only update VBN pointer)
;  2116          2167  2              !
;  2117          2168  2              ! 7) first pass of two-pass multibucket split case -
;  2118          2169  2              !    one of the new keys in each bucket:
;  2119          2170  2              !
;  2120          2171  2              !    size of first new key + VBN pointer size of bucket
;  2121          2172  2              !                          + number of new VBN bytes
;  2122          2173  2              !
;  2123          2174  2              ! 8) second pass of two-pass multibucket split case -
;  2124          2175  2              !    one of the new keys in each bucket:
;  2125          2176  2              !
;  2126          2177  2              !    size of second new key + VBN pointer size of bucket
;  2127          2178  2              !                           + number of new VBN bytes
;  2128          2179  2              !    (update one VBN pointer)
;  2129          2180  2              !
```

```
;  2130     2181  2          !  9) first pass of two-pass multibucket split case -
;  2131     2182  2          !     both of the new keys in the same bucket:
;  2132     2183  2          !
;  2133     2184  2          !     size of the two new keys + 2 * VBN pointer size of bucket
;  2134     2185  2          !                                + number of new VBN bytes
;  2135     2186  2          !
;  2136     2187  2          ! 10) second pass of two-pass multibucket split case -
;  2137     2188  2          !     both of the new keys in the same bucket:
;  2138     2189  2          !
;  2139     2190  2          !     number of new VBN bytes
;  2140     2191  2          !     (only update VBN pointer)
;  2141     2192  2          !
;  2142     2193  2          ! Two-pass Bucket Split Cases With Empty Buckets
;  2143     2194  2          !
;  2144     2195  2          ! 11) second pass of two-pass two-bucket split case:
;  2145     2196  2          !
;  2146     2197  2          !     number of new VBN bytes
;  2147     2198  2          !     (only update VBN pointer)
;  2148     2199  2          !
;  2149     2200  2          ! 12) first pass of two-pass multi-bucket split case:
;  2150     2201  2          !
;  2151     2202  2          !     size of new key + number of new VBN bytes
;  2152     2203  2          !     (update VBN pointer)
;  2153     2204  2          !
;  2154     2205  2          ! 13) second pass of two-pass multi-bucket split case:
;  2155     2206  2          !
;  2156     2207  2          !     number of new VBN bytes
;  2157     2208  2          !     (only update VBN pointer)
;  2158     2209  2          !
;  2159     2210  2          ! Sizes are always computed so as to include any control bytes
;  2160     2211  2          ! required. For cases 3, 4, and 9, key size include compression
;  2161     2212  2          ! of the second record if compression is enabled.
;  2162     2213  2          !
;  2163     2214  2          BEGIN
;  2164     2215  2          !
;  2165     2216  2          ! First set up some useful constants
;  2166     2217  2          !
;  2167     2218  2          LOCAL
;  2168     2219  2              CONTEXT : REF BBLOCK,   ! Address of block containing context
;  2169     2220  2                                      ! for potential update of an existing
;  2170     2221  2                                      ! index rec
;  2171     2222  2              KEY_SZ,                 ! Number of bytes needed to store the
;  2172     2223  2                                      ! key of the index record(s)
;  2173     2224  2                                      ! (compression index is added in and
;  2174     2225  2                                      ! front compression of the second
;  2175     2226  2                                      ! record is subtracted)
;  2176     2227  2              VBN_SZ,                 ! Size of all the VBNs after the
;  2177     2228  2                                      ! insertion (they could all grow by
;  2178     2229  2                                      ! one or two bytes)
;  2179     2230  2              NEW_VBN_BYTES;          ! Number of bytes which will be added
;  2180     2231  2                                      ! to the existing VBNs because of
;  2181     2232  2                                      ! changing to a larger VBN size
;  2182     2233  2                                      ! (may be 0)
;  2183     2234  2          !
;  2184     2235  2          ! Determine the size (VBN_SZ) of the VBNs after insertion of the
;  2185     2236  2          ! index records to be inserted, and the number of VBN bytes
;  2186     2237  3          ! (NEW_VBN_BYTES) which will have to be added inorder adjust the
```

```
;2187      2238  3          ! VBN pointer size of the pre-existing index records to this value.
;2188      2239  3          ! Determination of VBN_SZ will directly depend upon the bucket
;2189      2240  3          ! split case of the current invocation of this routine.
;2190      2241  3          !
;2191      2242  3          NEW_VBN_BYTES = 0;
;2192      2243  3
;2193      2244  3          ! For all one-pass bucket split cases and two-pass two-bucket
;2194      2245  3          ! empty bucket split cases the VBN pointer size is the maximum of
;2195      2246  3          ! the number of bytes need to store the current bucket pointer
;2196      2247  3          ! size, VBN_RIGHT, and VBN_MID.
;2197      2248  3          !
;2198      2249  4          IF (NOT .IRAB[IRB$V_SPL_IDX])
;2199      2250  3              AND
;2200      2251  4             (.IRAB[IRB$L_VBN_LEFT] NEQU 0)
;2201      2252  3          THEN
;2202      2253  3              VBN_SZ =  MAXU (RM$VBN_SIZE(.IRAB[IRB$L_VBN_MID]),
;2203      2254  3                             RM$VBN_SIZE(.IRAB[IRB$L_VBN_RIGHT]),
;2204      2255  3                             .BKT_ADDR[BKT$V_PTR_SZ]+2)
;2205      2256  3          ELSE
;2206      2257  3
;2207      2258  3              ! Determine VBN_SZ for the first pass of all two-pass
;2208      2259  3              ! bucket split cases.
;2209      2260  3              !
;2210      2261  3              IF .IRAB[IRB$V_SPL_IDX]
;2211      2262  3              THEN
;2212      2263  3
;2213      2264  3                  ! For case 9 (two-pass multi-bucket split with both keys
;2214      2265  3                  ! in the same bucket) and case 12 (multi-pass multi-bucket
;2215      2266  3                  ! empty bucket split) VBN_SZ is the maximum number of
;2216      2267  3                  ! bytes needed to store VBN_MID and the current bucket
;2217      2268  3                  ! VBN pointer size.
;2218      2269  3                  !
;2219      2270  3                  IF .IRAB[IRB$V_BIG_SPLIT]
;2220      2271  3                      AND
;2221      2272  5                     ((.IRAB[IRB$W_POS_INS] EQLU BKT$C_OVERHDSZ)
;2222      2273  4                                 OR
;2223      2274  4                                .IRAB[IRB$V_EMPTY_BKT])
;2224      2275  3                  THEN
;2225      2276  3                      VBN_SZ =  MAXU (RM$VBN_SIZE(.IRAB[IRB$L_VBN_MID]),
;2226      2277  3                                      .BKT_ADDR[BKT$V_PTR_SZ]+2)
;2227      2278  3
;2228      2279  3                  ! For all other two-pass bucket split cases, the VBN_SZ
;2229      2280  3                  ! is the number of bytes need to store the current VBN
;2230      2281  3                  ! pointer bucket size.
;2231      2282  3                  !
;2232      2283  3                  ELSE
;2233      2284  3                      VBN_SZ = .BKT_ADDR[BKT$V_PTR_SZ]+2
;2234      2285  3
;2235      2286  3          ! Determine VBN_SZ for the second pass of all two-pass
;2236      2287  3          ! bucket split cases.
;2237      2288  3          !
;2238      2289  3          ELSE
;2239      2290  3
;2240      2291  3              ! For case 8 (two-pass multibucket split with a key
;2241      2292  3              ! inserted in each of the buckets) VBN_SZ is the
;2242      2293  3              ! number of bytes needed to store the maximum of the
;2243      2294  3              ! current VBN bucket pointer size and VBN_MID.
```

```
;  2244     2295  3                    !
;  2245     2296  3                    IF .IRAB[IRB$L_VBN_MID] NEQU 0
;  2246     2297  3                    THEN
;  2247     2298  3                        VBN_SZ =  MAXU (RM$VBN_SIZE(.IRAB[IRB$L_VBN_MID]),
;  2248     2299  3                                        RM$VBN_SIZE(.IRAB[IRB$L_VBN_RIGHT]))
;  2249     2300  3
;  2250     2301  3                    ! For all other two-pass bucket split cases, VBN_SZ becomes
;  2251     2302  3                    ! the number of bytes needed to store VBN_RIGHT.
;  2252     2303  3                    !
;  2253     2304  3                    ELSE
;  2254     2305  3                        VBN_SZ = RM$VBN_SIZE(.IRAB[IRB$L_VBN_RIGHT]);
;  2255     2306  3
;  2256     2307  3                NEW_VBN_BYTES = RM$NEW_VBN_BYTES(.VBN_SZ);
;  2257     2308  3
;  2258     2309  3                ! Determine the key size of the first key to be inserted.
;  2259     2310  3                ! The two bytes required as overhead for key compression are
;  2260     2311  3                ! included in the key size if index compression is enabled.
;  2261     2312  3                !
;  2262     2313  3                IF .IDX_DFN [IDX$V_IDX_COMPR]
;  2263     2314  3                THEN
;  2264     2315  3                    KEY_SZ = 2
;  2265     2316  3                ELSE
;  2266     2317  3                    KEY_SZ = 0;
;  2267     2318  3
;  2268     2319  3                ! For all one-pass bucket split cases where there are keys to be
;  2269     2320  3                ! inserted, and for the first pass of all two-pass bucket split
;  2270     2321  3                ! cases, the first key to be inserted is the key found in key
;  2271     2322  3                ! buffer 2.
;  2272     2323  3                !
;  2273     2324  4                IF (.IRAB[IRB$L_VBN_LEFT] NEQU 0)
;  2274     2325  3                    AND
;  2275     2326  4                   (NOT (.IRAB[IRB$V_EMPTY_BKT] AND NOT .IRAB[IRB$V_BIG_SPLIT]))
;  2276     2327  3                THEN
;  2277     2328  3                    KEY_SZ = .KEY_SZ + RM$V3KEY_SZ (KEYBUF_ADDR(2))
;  2278     2329  3                ELSE
;  2279     2330  3
;  2280     2331  3                    ! For the second pass of the two-pass multibucket split case
;  2281     2332  3                    ! where a key goes in each of the two index buckets (case 8)
;  2282     2333  3                    ! the first and only key to be inserted is actually the second
;  2283     2334  3                    ! of the two keys to be inserted, and it is found in key
;  2284     2335  3                    ! buffer 3.
;  2285     2336  3                    !
;  2286     2337  3                    IF NOT .IRAB[IRB$V_EMPTY_BKT]
;  2287     2338  3                        AND
;  2288     2339  4                       (.IRAB[IRB$L_VBN_MID] NEQU 0)
;  2289     2340  3                    THEN
;  2290     2341  3                        KEY_SZ = .KEY_SZ + RM$V3KEY_SZ (KEYBUF_ADDR(3))
;  2291     2342  3
;  2292     2343  3                    ! For the second pass of all other two-pass split cases,
;  2293      344  3                    ! the key size is zero.
;  2294     2345  3                    !
;  2295     2346  3                    ELSE
;  2296     2347  3                        KEY_SZ = 0;
;  2297     2348  3
;  2298     2349  3                IF .IRAB [IRB$V_EMPTY_BKT]
;  2299     2350  3                THEN
;  2300     2351  3
```

```
;  2301          2352  3                ! We have an empty bucket.
;  2302          2353  3                !
;  2303          2354  4                BEGIN
;  2304          2355  4                SIZE = 0;
;  2305          2356  4
;  2306          2357  4                ! If there was a multiple bucket split, add new key and VBN
;  2307          2358  4                ! unless this is the second pass of a two-pass multi-bucket
;  2308          2359  4                ! empty bucket split case.
;  2309          2360  4                !
;  2310          2361  5                IF (.IRAB[IRB$V_BIG_SPLIT]
;  2311          2362  5                    AND
;  2312          2363  5                   (.IRAB[IRB$L_VBN_LEFT] NEQA 0))
;  2313          2364  4                THEN
;  2314          2365  5                    BEGIN
;  2315          2366  5                    SIZE = .KEY_SZ;
;  2316          2367  5
;  2317          2368  5                    ! If this is not a two-pass split case, the number of
;  2318          2369  5                    ! new VBN bytes is VBN size plus the number of bytes
;  2319          2370  5                    ! which will have to be added because of a possible
;  2320          2371  5                    ! VBN size change within the index bucket (will be
;  2321          2372  5                    ! zero if no VBN size change occurred).
;  2322          2373  5                    !
;  2323          2374  5                    IF NOT .IRAB[IRB$V_SPL_IDX]
;  2324          2375  5                    THEN
;  2325          2376  5                        NEW_VBN_BYTES = .NEW_VBN_BYTES + .VBN_SZ
;  2326          2377  5
;  2327          2378  5                    ! If this is the first pass of a two-pass multi-bucket
;  2328          2379  5                    ! empty bucket split case then the number of new VBN bytes
;  2329          2380  5                    ! will be the number of bytes which will have to be added
;  2330          2381  5                    ! because of a possible VBN size change within the index
;  2331          2382  5                    ! bucket provided we are allowed to update the VBN of the
;  2332          2383  5                    ! current record because it is the same as the VBN of the
;  2333          2384  5                    ! leftmost primary data bucket. If this is not the case,
;  2334          2385  5                    ! then because we can't update the VBN (this could cause
;  2335          2386  5                    ! crossed pointers down to the data level), the number
;  2336          2387  5                    ! of new VBN bytes is set to zero.
;  2337          2388  5                    !
;  2338          2389  5                    ELSE
;  2339          2390  6                        IF (.IRAB[IRB$L_VBN_LEFT] NEQA RM$V3_VBN())
;  2340          2391  5                        THEN
;  2341          2392  5                            NEW_VBN_BYTES = 0;
;  2342          2393  5                    END
;  2343          2394  5
;  2344          2395  5                ! If there wasn't a multibucket split, and we are not going
;  2345          2396  5                ! to swing a pointer because this could be causing crossed
;  2346          2397  5                ! pointers down to the level below, then there is need to add
;  2347          2398  5                ! new VBN bytes.
;  2348          2399  5                !
;  2349          2400  4                ELSE
;  2350          2401  5                    IF  (NOT .IRAB[IRB$V_BIG_SPLIT]
;  2351          2402  5                         AND
;  2352          2403  5                        (.IRAB [IRB$L_VBN_LEFT] NEQU RM$V3_VBN()))
;  2353          2404  4                    THEN
;  2354          2405  4                        NEW_VBN_BYTES = 0;
;  2355          2406  4
;  2356          2407  4                END
;  2357          2408  4
```

```
;  2358     2409  3              ELSE
;  2359     2410  3
;  2360     2411  3                  ! No empty buckets. For now, assume that entire record
;  2361     2412  3                  ! will fit into one bucket and compute its size.
;  2362     2413  3                  ! Always add VBN and key unless this is the second pass of one
;  2363     2414  3                  ! of the two-pass bucket split cases where all we are going
;  2364     2415  3                  ! to do is update a VBN pointer. In such cases, the size
;  2365     2416  3                  ! returned does not include the size of a VBN and a key.
;  2366     2417  3                  !
;  2367     2418  4                  BEGIN
;  2368     2419  4                  SIZE = .KEY_SZ;
;  2370     2420  4
;  2371       421  5                  IF (.IRAB[IRB$L_VBN_LEFT] NEQU 0)
;  2372       422  4                      OR
;  2373     2423  5                     (.IRAB[IRB$L_VBN_MID] NEQU 0)
;  2374     2424  4                  THEN
;  2375     2425  4                      NEW_VBN_BYTES = .NEW_VBN_BYTES + .VBN_SZ;
;  2376     2426  4
;  2377     2427  4                  ! If this is a one-pass multibucket split or the first pass
;  2378     2428  4                  ! of a two-pass multibucket split where both index records
;  2379     2429  4                  ! are to go in the same (old) index bucket, the size of the
;  2380     2430  4                  ! index record must include (in addition to the key size of the
;  2381     2431  4                  ! first index record, the new VBN size of the bucket, and the
;  2382     2432  4                  ! number of new VBN bytes which must be added to all
;  2383     2433  4                  ! pre-existing VBN pointers to make them all the same size) the
;  2384     2434  4                  ! key size of the second index record (less its front
;  2385     2435  4                  ! compression if index compression is enabled) and the size of
;  2386     2436  4                  ! another new VBN. If index compression is enabled, then the
;  2387     2437  4                  ! key size will not include any rear-end truncated characters.
;  2388     2438  4                  !
;  2389     2439  4                  IF .IRAB[IRB$V_BIG_SPLIT]
;  2390     2440  4                     AND
;  2391     2441  6                          ((NOT .IRAB[IRB$V_SPL_IDX]
;  2392     2442  6                                  AND .IRAB[IRB$L_VBN_LEFT] NEQU 0)
;  2393     2443  5                           OR
;  2394     2444  6                          (.IRAB[IRB$V_SPL_IDX]
;  2395     2445  5                                  AND .IRAB[IRB$W_POS_INS] EQLU BKT$C_OVERHDSZ))
;  2396     2446  4                  THEN
;  2397     2447  5                      BEGIN
;  2398     2448  5                      NEW_VBN_BYTES = .NEW_VBN_BYTES + .VBN_SZ;
;  2399     2449  5
;  2400     2450  5                      IF NOT .IDX_DFN [IDX$V_IDX_COMPR]
;  2401     2451  5                      THEN
;  2402     2452  5                          SIZE = .SIZE + .IDX_DFN [IDX$B_KEYSZ]
;  2403     2453  5                      ELSE
;  2404     2454  6                          BEGIN
;  2405     2455  6
;  2406     2456  6                          LOCAL
;  2407     2457  6                              KEYBUF3     : REF BBLOCK,
;  2408     2458  6                              KEYBUF4     : REF BBLOCK,
;  2409     2459  6                              KEYBUF5     : REF BBLOCK,
;  2410     2460  6                              SAVE_BKT_ADDR;
;  2411     2461  6
;  2412     2462  6                          ! To determine the size of both keys, we will
;  2413     2463  6                          ! need key buffers 3, 4, and 5.
;  2414     2464  6                          !
;  2415     2465  6                          KEYBUF3 = KEYBUF_ADDR(3);
```

```
;  2415          2466  6                    KEYBUF4 = KEYBUF_ADDR(4);
;  2416          2467  6                    KEYBUF5 = KEYBUF_ADDR(5);
;  2417          2468  6
;  2418          2469  6                    ! Move the second key into key buffer 4 and compress
;  2419          2470  6                    ! it as if it was to go at the point of insertion
;  2420          2471  6                    ! instead of following the first key (which is to
;  2421          2472  6                    ! be inserted there). Then recompress the second key
;  2422          2473  6                    ! based upon the first key (which is still in key
;  2423          2474  6                    ! buffer 5).
;  2424          2475  6                    !
;  2425          2476  6                    RM$MOVE (.KEYBUF3[KEY_LEN] + 2,
;  2426          2477  6                             .KEYBUF3,
;  2427          2478  6                             .KEYBUF4);
;  2428          2479  6                    RM$COMPRESS_KEY (.KEYBUF4);
;  2429          2480  6
;  2430          2481  6                    SAVE_BKT_ADDR = .BKT_ADDR;
;  2431          2482  6                    BKT_ADDR = 0;
;  2432          2483  6                    RM$RECOMPR_KEY (.KEYBUF5, .KEYBUF4);
;  2433          2484  6                    BKT_ADDR = .SAVE_BKT_ADDR;
;  2434          2485  6
;  2435          2486  6                    ! Add the size of the second key to the size of the
;  2436          2487  6                    ! first key to obtain the key contribution to the
;  2437          2488  6                    ! size of the record.
;  2438          2489  6                    !
;  2439          2490  6                    SIZE = .SIZE + .KEYBUF4[KEY_LEN] + 2;
;  2440          2491  5                    END;
;  2441          2492  4                END;
;  2442          2493  3            END;
;  2443          2494  3
;  2444          2495  3            ! Return the size of the record for prolog 3 index buckets as two
;  2445          2496  3            ! contigious words. The high order word contains the number of VBN
;  2446          2497  3            ! bytes required while the low order word contains the number of
;  2447          2498  3            ! bytes required to hold the key(s).
;  2448          2499  3            !
;  2449          2500  3            RETURN (.NEW_VBN_BYTES)^16 + .SIZE;
;  2450          2501  2            END;
;  2451          2502  2
;  2452          2503  2        ! Return the size of the prologue 1 & 2 record or Pprologue 3 SIDR as
;  2453          2504  2        ! a single unified quantity.
;  2454          2505  2        !
;  2455          2506  2        RETURN .SIZE;
;  2456          2507  1        END;


                                 1C  BB 00000 RM$RECORD_SIZE::
                                              PUSHR   #^M<R2,R3,R4>                   ; 1772
                              5E                 0C  C2 00002          SUBL2   #12, SP
                                              41 A9  95 00005          TSTB    65(IRAB)                ; 1854
                                                 5F  12 00008          BNEQ    8$
                                              78 A9  DD 0000A          PUSHL   120(IRAB)               ; 1861
                                                 0000V 30 0000D        BSBW    RM$VBN_SIZE
                              5E                 04  C0 00010          ADDL2   #4, SP
                              52                 50  D0 00013          MOVL    R0, SIZE
                                                 50  D4 00016          CLRL    R0                      ; 1863
```

```
                        03     00B7   CA  91 00018          CMPB    183(IFAB), #3                                    ;
                                      07  1E 0001D          BGEQU   1$                                               ;
                                      50  D6 0001F          INCL    R0                                               ;
                        52            02  C0 00021          ADDL2   #2, SIZE                                         ; 1865
                                      03  11 00024          BRB     2$                                               ;
                        52            03  C0 00026 1$:      ADDL2   #3, SIZE                                         ; 1867
                              44      A9  95 00029 2$:      TSTB    68(IRAB)                                         ; 1873
                                      39  19 0002C          BLSS    7$                                               ;
                        1A            50  E9 0002E          BLBC    R0, 6$                                           ; 1878
                        04      1C    A7  E9 00031          BLBC    28(IDX_DFN), 3$                                  ; 1888
                  05        44  A9        04  E0 00035      BBS     #4, 68(IRAB), 3$                                 ; 1890
                        52            08  C0 0003A          ADDL2   #8, SIZE                                         ; 1893
                                      03  11 0003D          BRB     4$                                               ; 1892
                        52            04  C0 0003F 3$:      ADDL2   #4, SIZE                                         ; 1895
                        50      20    A7  9A 00042 4$:      MOVZBL  32(IDX_DFN), R0                                  ; 1897
                        52            50  C0 00046 5$:      ADDL2   R0, SIZE                                         ;
                                      1C  11 00049          BRB     7$                                               ; 1878
                        50     00B4   CA  3C 0004B 6$:      MOVZWL  180(IFAB), R0                                    ; 1904
                              60 B940  9F  00 00050          PUSHAB  @96(IRAB)[R0]                                    ;
                                0000V 30  00054              BSBW    RM$V3KEY_SZ                                      ;
                        5E            04  C0 00057          ADDL2   #4, SP                                           ;
                        52      02 A042  9E 0005A          MOVAB   2(R0)[SIZE], SIZE                                ;
                  6D        1C  A7        06  E1 0005F      BBC     #6, 28(IDX_DFN), 12$                             ; 1906
                        52            02  C0 00064          ADDL2   #2, SIZE                                         ; 1908
                                      68  11 00067 7$:      BRB     12$                                              ; 1878
                        03     00B7   CA  91 00069 8$:      CMPB    183(IFAB), #3                                    ; 1914
                                      03  1F 0006E          BLSSU   9$                                               ;
                                    00E2  31 00070          BRW     21$                                              ;
                                      52  D4 00073 9$:      CLRL    SIZE                                             ; 1999
                  73        44  A9        06  E1 00075      BBC     #6, 68(IRAB), 14$                                ; 2003
                        5C            01  D0 0007A          MOVL    #1, AP                                           ; 2006
                                    0000G 30  0007D          BSBW    RM$RECORD_VBN                                    ; 2013
                        50     0088   C9  D1 00080          CMPL    136(IRAB), R0                                    ;
                                      29  12 00085          BNEQ    11$                                              ;
                        0F      44    A9  E9 00087          BLBC    68(IRAB), 10$                                    ; 2020
                              0090   C9  DD 0008B          PUSHL   144(IRAB)                                        ; 2022
                                    0000V 30  0008F          BSBW    RM$VBN_SIZE                                      ;
                        5E            04  C0 00092          ADDL2   #4, SP                                           ;
                        52            50  D0 00095          MOVL    R0, SIZE                                         ;
                                      16  11 00098          BRB     11$                                              ;
                              0090   C9  DD 0009A 10$:      PUSHL   144(IRAB)                                        ; 2030
                                    0000V 30  0009E          BSBW    RM$VBN_SIZE                                      ;
                        5E            04  C0 000A1          ADDL2   #4, SP                                           ;
  51        66        02            00  EF 000A4          EXTZV   #0, #2, (REC_ADDR), R1                           ; 2031
                        50            51  C2 000A9          SUBL2   R1, R0                                           ;
                        52      FE    A0  9E 000AC          MOVAB   -2(R0), SIZE                                     ;
                  1C        44  A9        02  E1 000B0 11$:      BBC     #2, 68(IRAB), 12$                            ; 2033
                              0088   C9  D5 000B5          TSTL    136(IRAB)                                        ; 2043
                                      18  12 000B9          BNEQ    13$                                              ;
                              008C   C9  DD 000BB          PUSHL   140(IRAB)                                        ; 2045
                                    0000V 30  000BF          BSBW    RM$VBN_SIZE                                      ;
                        5E            04  C0 000C2          ADDL2   #4, SP                                           ;
  51        66        02            00  EF 000C5          EXTZV   #0, #2, (REC_ADDR), R1                           ; 2046
                        50            51  C2 000CA          SUBL2   R1, R0                                           ;
                        52      FE    A0  9E 000CD          MOVAB   -2(R0), SIZE                                     ;
                                      7F  11 000D1 12$:      BRB     20$                                              ; 2045
                        50      20    A7  9A 000D3 13$:      MOVZBL  32(IDX_DFN), R0                                  ; 2056
```

```
                                52      01 A042  9E 000D7          MOVAB   1(R0)[SIZE], SIZE                          ;
                                72      44   A9  E8 000DC          BLBS    68(IRAB), 20$                               ; 2058
                              008C      C9  DD 000E0              PUSHL   140(IRAB)                                   ; 2060
                                    0000V 30 000E4              BSBW    RMSVBN_SIZE                                 ;
                                5E        04  C0 000E7          ADDL2   #4, SP                                      ;
                                    FF59  31 000EA              BRW     5$                                          ;
                                16      44   A9  E8 000ED 14$:  BLBS    68(IRAB), 15$                               ; 2080
                              008C      C9  DD 000F1              PUSHL   140(IRAB)                                   ; 2082
                                    0000V 30 000F5              BSBW    RMSVBN_SIZE                                 ;
                                5E        04  C0 000F8          ADDL2   #4, SP                                      ;
        51          66          02        00  EF 000FB          EXTZV   #0, #2, (REC_ADDR), R1                      ; 2083      ; R
                                50        51  C2 00100          SUBL2   R1, R0                                      ;
                                52      FE   A0  9E 00103          MOVAB   -2(R0), SIZE                            ;
                    0A      44  A9        02  E1 00107 15$:      BBC     #2, 68(IRAB), 16$                           ; 2094
                                0A      44   A9  E8 0010C          BLBS    68(IRAB), 17$                               ;
                              0090      C9  D5 00110              TSTL    144(IRAB)                                   ; 2095
                                        0F  12 00114              BNEQ    18$                                         ;
                                21      44   A9  E9 00116 16$:      BLBC    68(IRAB), 19$                           ; 2097
                    1C      44  A9        02  E1 0011A 17$:      BBC     #2, 68(IRAB), 19$                           ; 2098
                                0E      48   A9  B1 0011F          CMPW    72(IRAB), #14                               ; 2099
                                        16  12 00123              BNEQ    19$                                         ;
                              0090      C9  DD 00125 18$:          PUSHL   144(IRAB)                                   ; 2101
                                    0000V 30 00129              BSBW    RMSVBN_SIZE                                 ;
                                5E        04  C0 0012C          ADDL2   #4, SP                                      ;
                                50        52  C0 0012F          ADDL2   SIZE, R0                                    ;
                                51      20   A7  9A 00132          MOVZBL  32(IDX_DFN), R1                         ; 2102
                                52      01 A140  9E 00136          MOVAB   1(R1)[R0], SIZE                         ;
                              0088      C9  D5 0013B 19$:          TSTL    136(IRAB)                                   ; 2118
                                        11  13 0013F              BEQL    20$                                         ;
        50          66          02        00  EF 00141          EXTZV   #0, #2, (REC_ADDR), R0                      ; 2120
                                50        52  C0 00146          ADDL2   SIZE, R0                                    ;
                                51      20   A7  9A 00149          MOVZBL  32(IDX_DFN), R1                         ; 2121
                                52      03 A140  9E 0014D          MOVAB   3(R1)[R0], SIZE                         ;
                                    01C7  31 00152 20$:          BRW     51$                                         ; 1914
                                08      AE  D4 00155 21$:          CLRL    NEW_VBN_BYTES                               ; 2242
                                37      44   A9  E8 00158          BLBS    68(IRAB), 24$                               ; 2249
                              0088      C9  D5 0015C              TSTL    136(IRAB)                                   ; 2251
                                        2D  13 00160              BEQL    23$                                         ;
                              0090      C9  DD 00162              PUSHL   144(IRAB)                                   ; 2253
                                    0000V 30 00166              BSBW    RMSVBN_SIZE                                 ;
                                51        50  D0 00169          MOVL    R0, R1                                      ;
                                6E    008C      C9  D0 0016C          MOVL    140(IRAB), (SP)                         ; 2254
                                    0000V 30 00171              BSBW    RMSVBN_SIZE                                 ;
                                5E        04  C0 00174          ADDL2   #4, SP                                      ;
                                50        51  D1 00177          CMPL    R1, R0                                      ;
                                        03  1E 0017A              BGEQU   22$                                         ;
                                51        50  D0 0017C          MOVL    R0, R1                                      ;
        50      0D  A5          02        03  EF 0017F 22$:      EXTZV   #3, #2, 13(BKT_ADDR), R0                    ; 2255
                                50        02  C0 00185          ADDL2   #2, R0                                      ;
                                50        51  D1 00188          CMPL    R1, R0                                      ;
                                        5F  1F 0018B              BLSSU   28$                                         ;
                                        60  11 0018D              BRB     29$                                         ; 2253
                                3A      44   A9  E9 0018F 23$:      BLBC    68(IRAB), 27$                           ; 2261
                    28      44  A9        02  E1 00193 24$:      BBC     #2, 68(IRAB), 26$                           ; 2270
                                0E      48   A9  B1 00198          CMPW    72(IRAB), #14                               ; 2272
                                        05  13 0019C              BEQL    25$                                         ;
                    1D      44  A9        06  E1 0019E          BBC     #6, 68(IRAB), 26$                           ; 2274
```

```
                                0090   C9  DD 001A3 25$:    PUSHL   144(IRAB)                                ; 2276
                                     0000V 30 001A7         BSBW    RM$VBN_SIZE                             ;
                                  5E   04  C0 001AA         ADDL2   #4, SP                                  ;
          51        0D   A5       02   03  EF 001AD         EXTZV   #3, #2, 13(BKT_ADDR), R1                ; 2277
                                  51   02  C0 001B3         ADDL2   #2, R1                                  ;
                                  51   50  D1 001B6         CMPL    R0, R1                                  ;
                                       44  1E 001B9         BGEQU   31$                                     ;
                                  50   51  D0 001BB         MOVL    R1, R0                                  ;
                                       3F  11 001BE         BRB     31$                                     ; 2276
    04    AE        0D   A5       02   03  EF 001C0 26$:    EXTZV   #3, #2, 13(BKT_ADDR), VBN_SZ            ; 2284
                04    AE         02   C0 001C7         ADDL2   #2, VBN_SZ                                  ;
                                       36  11 001CB         BRB     32$                                     ; 2270
                                  50   0090  C9  D0 001CD 27$:    MOVL    144(IRAB), R0                   ; 2296
                                       21  13 001D2         BEQL    30$                                     ;
                                       50  DD 001D4         PUSHL   R0                                      ; 2298
                                     0000V 30 001D6         BSBW    RM$VBN_SIZE                             ;
                                  51   50  D0 001D9         MOVL    R0, R1                                  ;
                                  6E 008C  C9  D0 001DC     MOVL    140(IRAB), (SP)                         ; 2299
                                     0000V 30 001E1         BSBW    RM$VBN_SIZE                             ;
                                  5E   04  C0 001E4         ADDL2   #4, SP                                  ;
                                  50   51  D1 001E7         CMPL    R1, R0                                  ;
                                       03  1E 001EA         BGEQU   29$                                     ;
                                  51   50  D0 001EC 28$:    MOVL    R0, R1                                  ;
                04    AE         51   D0 001EF 29$:         MOVL    R1, VBN_SZ                              ; 2298
                                       0E  11 001F3         BRB     32$                                     ;
                                  008C  C9  DD 001F5 30$:    PUSHL   140(IRAB)                               ; 2305
                                     0000V 30 001F9         BSBW    RM$VBN_SIZE                             ;
                                  5E   04  C0 001FC         ADDL2   #4, SP                                  ;
                04    AE         50   D0 001FF 31$:         MOVL    R0, VBN_SZ                              ;
                          04    AE   DD 00203 32$:         PUSHL   VBN_SZ                                  ; 2307
                                FDB6   30 00206             BSBW    RM$NEW_VBN_BYTES                        ;
                                  5E   04  C0 00209         ADDL2   #4, SP                                  ;
                08    AE         50   D0 0020C             MOVL    R0, NEW_VBN_BYTES                       ;
    05          1C    A7         03   E1 00210             BBC     #3, 28(IDX_DFN), 33$                    ; 2313
                                  6E   02  D0 00215         MOVL    #2, KEY_SZ                              ; 2315
                                       02  11 00218         BRB     34$                                     ;
                                  6E   D4  0021A 33$:       CLRL    KEY_SZ                                  ; 2317
                                  0088  C9  D5 0021C 34$:    TSTL    136(IRAB)                               ; 2324
                                       15  13 00220         BEQL    36$                                     ;
    05          44    A9         06   E1 00222             BBC     #6, 68(IRAB), 35$                       ; 2326
    0B          44    A9         02   E1 00227             BBC     #2, 68(IRAB), 36$                       ;
                                  50   00B4  CA  3C 0022C 35$:    MOVZWL  180(IFAB), R0                   ; 2328
                                    60 B940  9F 00231         PUSHAB  @96(IRAB)[R0]                           ;
                                       14  11 00235         BRB     37$                                     ;
    1A          44    A9         06   E0 00237 36$:         BBS     #6, 68(IRAB), 38$                       ; 2337
                                  0090  C9  D5 0023C         TSTL    144(IRAB)                               ; 2339
                                       14  13 00240         BEQL    38$                                     ;
                                  50   00B4  CA  3C 00242     MOVZWL  180(IFAB), R0                   ; 2341
                                    60 B940  3F 00247         PUSHAW  @96(IRAB)[R0]                           ;
                                     0000V 30 0024B 37$:      BSBW    RM$V3KEY_SZ                             ;
                                  5E   04  C0 0024E         ADDL2   #4, SP                                  ;
                                  6E   50  C0 00251         ADDL2   R0, KEY_SZ                              ;
                                       02  11 00254         BRB     39$                                     ;
                                  6E   D4  00256 38$:       CLRL    KEY_SZ                                  ; 2347
                                  53   44  A9  9E 00258 39$:    MOVAB   68(IRAB), R3                    ; 2349
    2C                            63   06  E1 0025C         BBC     #6, (R3), 42$                           ;
                                       52  D4 00260         CLRL    SIZE                                    ; 2355
```

```
                17          63                  02  E1 00262          BBC     #2, (R3), 41$                        ; 2361
                                          0088  C9  D5 00266          TSTL    136(IRAB)                            ; 2363
                                                0D  13 0026A          BEQL    40$                                  ;
                            52                  6E  D0 0026C          MOVL    KEY_SZ, SIZE                         ; 2366
                            0B                  63  E8 0026F          BLBS    (R3), 41$                            ; 2374
                        08  AE              04  AE  C0 00272          ADDL2   VBN_SZ, NEW_VBN_BYTES                ; 2376
                                                51  11 00277          BRB     48$                                  ;
                4D          63                  02  E0 00279 40$:     BBS     #2, (R3), 48$                        ; 2401
                                              0000G 30 0027D 41$:     BSBW    RM$V3_VBN                            ; 2403
                            50              0088  C9  D1 00280          CMPL    136(IRAB), R0                        ;
                                                43  13 00285          BEQL    48$                                  ;
                                            08  AE  D4 00287          CLRL    NEW_VBN_BYTES                        ; 2405
                                                3E  11 0028A          BRB     48$                                  ; 2349
                            52                  6E  D0 0028C 42$:     MOVL    KEY_SZ, SIZE                         ; 2419
                                                50  D4 0028F          CLRL    R0                                   ; 2421
                                          0088  C9  D5 00291          TSTL    136(IRAB)                            ;
                                                04  13 00295          BEQL    43$                                  ;
                                                50  D6 00297          INCL    R0                                   ;
                                                06  11 00299          BRB     44$                                  ;
                                          0090  C9  D5 0029B 43$:     TSTL    144(IRAB)                            ; 2423
                                                05  13 0029F          BEQL    45$                                  ;
                        08  AE              04  AE  C0 002A1 44$:     ADDL2   VBN_SZ, NEW_VBN_BYTES                ; 2425
                68          63                  02  E1 002A6 45$:     BBC     #2, (R3), 50$                        ; 2439
                            06                  63  E8 002AA          BLBS    (R3), 46$                            ; 2441
                            09                  50  E8 002AD          BLBS    R0, 47$                              ; 2442
                            5F                  63  E9 002B0          BLBC    (R3), 50$                            ; 2444
                            0E              48  A9  B1 002B3 46$:     CMPW    72(IRAB), #14                        ; 2445
                                                59  12 002B7          BNEQ    50$                                  ;
                        08  AE              04  AE  C0 002B9 47$:     ADDL2   VBN_SZ, NEW_VBN_BYTES                ; 2448
            09          1C  A7                  03  E0 002BE          BBS     #3, 28(IDX_DFN), 49$                 ; 2450
                            50              20  A7  9A 002C3          MOVZBL  32(IDX_DFN), R0                      ; 2452
                            52                  50  C0 002C7          ADDL2   R0, SIZE                             ;
                                                46  11 002CA 48$:     BRB     50$                                  ;
                            51          00B4    CA  3C 002CC 49$:     MOVZWL  180(IFAB), R1                        ; 2465
                            50              60 B941 3E 002D1          MOVAW   @96(IRAB)[R1], KEYBUF3               ;
                53          51                  03  C5 002D6          MULL3   #3, R1, R3                           ; 2466
                            53              60  A9  C0 002DA          ADDL2   96(IRAB), KEYBUF4                    ;
                        04  AE              60 B941 DE 002DE          MOVAL   @96(IRAB)[R1], KEYBUF5               ; 2467
                                                09  BB 002E4          PUSHR   #^M<R0,R3>                           ; 2477
                            7E                  60  9A 002E6          MOVZBL  (KEYBUF3), -(SP)                     ; 2476
                            6E                  02  C0 002E9          ADDL2   #2, (SP)                             ;
                                              0000G 30 002EC          BSBW    RM$MOVE                              ;
                            5E                  0C  C0 002EF          ADDL2   #12, SP                              ;
                            50                  53  D0 002F2          MOVL    KEYBUF4, R0                          ; 2479
                                              0000G 30 002F5          BSBW    RM$COMPRESS_KEY                      ;
                            54                  55  D0 002F8          MOVL    BKT_ADDR, SAVE_BKT_ADDR              ; 2481
                                                55  D4 002FB          CLRL    BKT_ADDR                             ; 2482
                            51                  53  D0 002FD          MOVL    KEYBUF4, R1                          ; 2483
                            50              04  AE  D0 00300          MOVL    KEYBUF5, R0                          ;
                                              0000G 30 00304          BSBW    RM$RECCMPR_KEY                       ;
                            55                  54  D0 00307          MOVL    SAVE_BKT_ADDR, BKT_ADDR              ; 2484
                            50                  63  9A 0030A          MOVZBL  (KEYBUF4), R0                        ; 2490
                            52              02 A042 9E 0030D          MOVAB   2(R0)[SIZE], SIZE                    ;
                50      08  AE                  10  78 00312 50$:     ASHL    #16, NEW_VBN_BYTES, R0               ; 2500
                            50                  52  C0 00317          ADDL2   SIZE, R0                             ;
                                                03  11 0031A          BRB     52$                                  ;
                            50                  52  D0 0031C 51$:     MOVL    SIZE, R0                             ; 2506
```

```
                          5E              0C  C0 0031F 52$:    ADDL2   #12, SP                          ; 2507
                                          1C  BA 00322          POPR    #^M<R2,R3,R4>                    ;
                                              05 00324          RSB                                      ;

; Routine Size:  805 bytes,    Routine Base:  RM$RMS3 + 04F5
```

```
;2458    2508  1 %SBTTL  'RM$SHFT_VBNS'
;2459    2509  1 ROUTINE RM$SHFT_VBNS : RL$COMMON_LINK NOVALUE =
;2460    2510  1
;2461    2511  1 !++
;2462    2512  1 !
;2463    2513  1 !   RM$SHFT_VBNS
;2464    2514  1 !
;2465    2515  1 !       Spread the VBNs at the end of the bucket to make room for
;2466    2516  1 !       the new VBN(s).
;2467    2517  1 !
;2468    2518  1 !       This routine handles the following conditions:
;2469    2519  1 !
;2470    2520  1 !               1) two bucket split, no empty buckets
;2471    2521  1 !               2) two bucket split, empty bucket
;2472    2522  1 !               3) multi-bucket split (three or four), no empty buckets
;2473    2523  1 !               4) multi-hucket split, empty buckets
;2474    2524  1 !
;2475    2525  1 !       The two-pass non-empty bucket bucket split cases which occur when the
;2476    2526  1 !       insertion point is found to be the same as the bucket split point are
;2477    2527  1 !       handled as follows:
;2478    2528  1 !
;2479    2529  1 !               5) first pass two bucket split    - treated as 1)
;2480    2530  1 !               6) second pass two bucket split   - treated as 2)
;2481    2531  1 !                  first pass multi-bucket split
;2482    2532  1 !               7)      one in each bucket        - treated as 4)
;2483    2533  1 !               8)      both in old bucket        - treated as 3)
;2484    2534  1 !                  second pass multi-bucket split
;2485    2535  1 !               9)      one in each bucket        - treated as 4)
;2486    2536  1 !              10)      both in old bucket        - treated as 2)
;2487    2537  1 !
;2488    2538  1 !       The two-pass empty bucket bucket split cases which occur when the
;2489    2539  1 !       insertion point is found to be the same as the bucket split point are
;2490    2540  1 !       handled as follows:
;2491    2541  1 !
;2492    2542  1 !               11) second pass two bucket split   - treated as 2)
;2493    2543  1 !               12) first pass multi-bucket split  - treated as 4)
;2494    2544  1 !               13) second pass multi-bucket split - treated as 2)
;2495    2545  1 !
;2496    2546  1 !       In all cases it assumes that the new VBNs will be the same size
;2497    2547  1 !       as the current VBNs. If they actually are larger, the routine that
;2498    2548  1 !       inserts them will have to grow each one individually.
;2499    2549  1 !
;2500    2550  1 !   INPUT
;2501    2551  1 !
;2502    2552  1 !       none
;2503    2553  1 !
;2504    2554  1 !   IMPLICIT INPUT
;2505    2555  1 !
;2506    2556  1 !       IRAB
;2507    2557  1 !           [EMPTY_BUCKET]
;2508    2558  1 !           [BIG_SPLIT]
;2509    2559  1 !           [VBN_LEFT]
;2510    2560  1 !           [VBN_MID]
;2511    2561  1 !           [SPL_IDX]
;2512    2562  1 !           [POS_INS]
;2513    2563  1 !       BKT_ADDR
;2514    2564  1 !
```

```
;2515     2565  1 !    OUTPUT
;2516     2566  1 !
;2517     2567  1 !       none
;2518     2568  1 !
;2519     2569  1 !    IMPLICIT OUTPUT
;2520     2570  1 !
;2521     2571  1 !       The VBN chain is spread apart and ready to insert new VBNs
;2522     2572  1 !
;2523     2573  1 !---
;2524     2574  1
;2525     2575  2     BEGIN
;2526     2576  2
;2527     2577  2     EXTERNAL REGISTER
;2528     2578  2         R_REC_ADDR_STR,
;2529     2579  2         R_BKT_ADDR_STR,
;2530     2580  2         R_IRAB_STR,
;2531     2581  2         R_IDX_DFN_STR,
;2532     2582  2         R_IFAB_STR;
;2533     2583  2
;2534     2584  2     GLOBAL REGISTER
;2535     2585  2         R_RAB,
;2536     2586  2         R_BDB,
;2537     2587  2         R_IMPURE;
;2538     2588  2
;2539     2589  2     MACRO
;2540     2590  2         OFFSET = 0,0,16,0 %;
;2541     2591  2
;2542     2592  2     LOCAL
;2543     2593  2         FREE_SPACE : REF BBLOCK,! Address of offset to VBN freespace
;2544     2594  2         LENGTH,                 ! Number of bytes to move
;2545     2595  2         SWING_FLG,              ! Logical - so we don't have to check swing conditions
;2546     2596  2                                 !           every time around
;2547     2597  2         INS_ADDR,               ! Address where the new VBN will be inserted
;2548     2598  2         NUM_BYTES;              ! Number of bytes which will be inserted
;2549     2599  2
;2550     2600  2     ! The swing flag is set under the following bucket split cases:
;2551     2601  2     !
;2552     2602  2     !    1. All empty bucket split cases
;2553     2603  2     !    2. First pass multibucket split cases where one key goes
;2554     2604  2     !       in each of the two index buckets
;2555     2605  2     !    3. All second pass bucket split cases
;2556     2606  2     !
;2557     2607  2     SWING_FLG = 0;
;2558     2608  2
;2559     2609  2     IF .IRAB[IRB$V_EMPTY_BKT]
;2560     2610  2         OR
;2561     2611  3         (.IRAB[IRB$L_VBN_LEFT] EQLU 0)
;2562     2612  2         OR
;2563     2613  3         (.IRAB[IRB$V_SPL_IDX]
;2564     2614  3                 AND .IRAB[IRB$V_BIG_SPLIT]
;2565     2615  3                 AND .IRAB[IRB$W_POS_INS] NEQU BKT$C_OVERHDSZ)
;2566     2616  2     THEN
;2567     2617  2         SWING_FLG = 1;
;2568     2618  2
;2569     2619  2     ! Assume a two bucket split with no empties
;2570     2620  2     !
;2571     2621  2     INS_ADDR = RM$CNTRL_ADDR();
```

```
;  2572     2622  2     NUM_BYTES = .BKT_ADDR [BKT$V_PTR_SZ] + 2;
;  2573     2623  2
;  2574     2624  2     ! Consider all nonmulti-bucket cases, the second pass of the multibucket
;  2575     2625  2     ! case where both index records go in the same (old) index bucket, and the
;  2576     2626  2     ! second pass of the multi-bucket with empty bucket split case.
;  2577     2627  2     !
;  2578     2628  3     IF (NOT .IRAB [IRB$V_BIG_SPLIT])
;  2579     2629  2         OR
;  2580     2630  3         (.IRAB[IRB$L_VBN_MID] EQLA 0)
;  2581     2631  2     THEN
;  2582     2632  3         BEGIN
;  2583     2633  3
;  2584     2634  3         IF .SWING_FLG
;  2585     2635  3         THEN
;  2586     2636  3
;  2587     2637  3             ! For cases 2, 6, 10, 11, and 13 - just update the VBN in place
;  2588     2638  3             !
;  2589     2639  3             NUM_BYTES = 0;
;  2590     2640  3
;  2591     2641  3         END
;  2592     2642  3
;  2593     2643  3     ! For cases 1, and 5 - shift the VBNs to make room for 1 new VBN
;  2594     2644  3     !
;  2595     2645  3     ! Consider all multibucket cases except case 10 and 13 handled above.
;  2596     2646  3     !
;  2597     2647  2     ELSE
;  2598     2648  2
;  2599     2649  2         IF NOT .SWING_FLG
;  2600     2650  2         THEN
;  2601     2651  2
;  2602     2652  2             ! For cases 3, and 8 - shift the VBNs to make room for 2 new VBNs
;  2603     2653  2             !
;  2604     2654  2             NUM_BYTES = .NUM_BYTES * 2;
;  2605     2655  2
;  2606     2656  2     ! For cases 4, 7, 9, and 12 - shift the VBNs to make room for 1 new VBN
;  2607     2657  2     ! Cases 4, and 9 will also update one VBN in place
;  2608     2658  2     !
;  2609     2659  2     ! Now actually shift the VBN chain provided at least one VBN is being added
;  2610     2660  2     ! and it is not being added to the end of the chain (ie the index record(s)
;  2611     2661  2     ! to be added do not have key values higher than all index records currently
;  2612     2662  2     ! in the bucket)
;  2613     2663  2     !
;  2614     2664  2     FREE_SPACE = .BKT_ADDR + (.IDX_DFN[IDX$B_IDXBKTSZ] * 512) - BKT$C_ENDOVHD;
;  2615     2665  2     LENGTH = .INS_ADDR - (.BKT_ADDR + .FREE_SPACE [OFFSET]);
;  2616     2666  2
;  2617     2667  3     IF (.LENGTH GTR 0)
;  2618     2668  2         AND
;  2619     2669  3         (.NUM_BYTES GTR 0)
;  2620     2670  2     THEN
;  2621     2671  2         RM$MOVE (.LENGTH, .BKT_ADDR + .FREE_SPACE [OFFSET],
;  2622     2672  2                  .BKT_ADDR + .FREE_SPACE [OFFSET] - .NUM_BYTES);
;  2623     2673  2
;  2624     2674  2     ! Update the free space pointer.
;  2625     2675  2     !
;  2626     2676  2     FREE_SPACE [OFFSET] = .FREE_SPACE [OFFSET] - .NUM_BYTES
;  2627     2677  2
;  2628     2678  1     END;
```

```
                                0914   8F   BB 00000 RM$SHFT_VBNS:
                                                     PUSHR   #^M<R2,R4,R8,R11>                       ; 2509
                                       54   D4 00004          CLRL    SWING_FLG                       ; 2607
                                       06   E0 00006          BBS     #6, 68(IRAB), 1$                ; 2609
                                0088   C9   D5 0000B          TSTL    136(IRAB)                       ; 2611
                                       0F   13 0000F          BEQL    1$                              ;
                                   0E  44   A9   E9 00011     BLBC    68(IRAB), 2$                    ; 2613
                                   09  44   A9   02 00015     BBC     #2, 68(IRAB), 2$                ; 2614
                                   0E  48   A9   B1 0001A     CMPW    72(IRAB), #14                   ; 2615
                                       03   13 0001E          BEQL    2$                              ;
                                   54       01   D0 00020 1$:  MOVL    #1, SWING_FLG                   ; 2617
                                          0000G 30 00023 2$:  BSBW    RM$CNTRL_ADDR                   ; 2621
                52  0D  A5             02   EF 00026          EXTZV   #3, #2, 3(BKT_ADDR), NUM_BYTES  ; 2622
                                   52       02   C0 0002C     ADDL2   #2, NUM_BYTES                   ;
                                   06  44   A9   02 0002F     BBC     #2, 68(IRAB), 3$                ; 2628
                                0090   C9   D5 00034          TSTL    144(IRAB)                       ; 2630
                                       07   12 00038          BNEQ    4$                              ;
                                   0A      54   E9 0003A 3$:  BLBC    SWING_FLG, 5$                   ; 2634
                                       52   D4 0003D          CLRL    NUM_BYTES                       ; 2639
                                       06   11 0003F          BRB     5$                              ; 2628
                                   03      54   E8 00041 4$:  BLBS    SWING_FLG, 5$                   ; 2649
                                   52      02   C4 00044     MULL2   #2, NUM_BYTES                   ; 2654
                                   51      16   A7   9A 00047 5$:  MOVZBL  22(IDX_DFN), R1            ; 2664
                51                 51      09   78 0004B     ASHL    #9, R1, R1                      ;
                                   51      FC A145  9E 0004F  MOVAB   -4(R1)[BKT_ADDR], FREE_SPACE    ;
                                   54      61   3C 00054     MOVZWL  (FREE_SPACE), R4                ; 2665
                                   54      55   C0 00057     ADDL2   BKT_ADDR, R4                    ;
                                   50      54   C2 0005A     SUBL2   R4, LENGTH                      ;
                                           10   15 0005D     BLEQ    6$                              ; 2667
                                           52   D5 0005F     TSTL    NUM_BYTES                       ; 2669
                                           0C   15 00061     BLEQ    6$                              ;
                            7E             54   52   C3 00063     SUBL3   NUM_BYTES, R4, -(SP)        ; 2672
                                           11   BB 00067     PUSHR   #^M<R0,R4>                      ; 2671
                                          0000G 30 00069     BSBW    RM$MOVE                         ;
                                   5E      0C   C0 0006C     ADDL2   #12, SP                         ;
                                   61      52   A2 0006F 6$:  SUBW2   NUM_BYTES, (FREE_SPACE)        ; 2676
                                0914   8F   BA 00072          POPR    #^M<R2,R4,R8,R11>               ; 2678
                                           05 00076          RSB                                     ;

; Routine Size:  119 bytes,    Routine Base:  RM$RMS3 + 081A

; 2629          2679  1
```

```
;2631    2680  1 %SBTTL  'RM$V3_IDX_REC'
;2632    2681  1 ROUTINE RM$V3_IDX_REC (REC_SZ) : RL$COMMON_LINK NOVALUE =
;2633    2682  1 !++
;2634    2683  1 !
;2635    2684  1 ! RM$V3_IDX_REC
;2636    2685  1 !
;2637    2686  1 !       This routine builds a new prologue three index record. It does
;2638    2687  1 !       one of the following
;2639    2688  1 !
;2640    2689  1 !               2) Change a VBN (swing pointer because of empty bucket)
;2641    2690  1 !               3) Add a key and VBN
;2642    2691  1 !               4) Add an additional key and VBN
;2643    2692  1 !
;2644    2693  1 ! CALLING SEQUENCE:
;2645    2694  1 !
;2646    2695  1 !       RM$V3_KEY_REC (REC_SZ)
;2647    2696  1 !
;2648    2697  1 ! INPUT PARAMETERS:
;2649    2698  1 !       REC_SZ  - total size of record to be inserted
;2650    2699  1 !
;2651    2700  1 ! IMPLICIT INPUTS:
;2652    2701  1 !       IRAB
;2653    2702  1 !       REC_ADDR
;2654    2703  1 !       BKT_ADDR
;2655    2704  1 !       IDX_DFN
;2656    2705  1 !       IRAB[IRB$V_SPL_IDX]
;2657    2706  1 !       IRAB[IRB$V_EMPTY_BKT]
;2658    2707  1 !       IRAB[IRB$B_BIG_SPLIT]
;2659    2708  1 !       IRAB[IRB$W_POS_INS]
;2660    2709  1 !       IRAB[IRB$L_VBN_LEFT]
;2661    2710  1 !       IRAB[IRB$L_VBN_MID]
;2662    2711  1 !
;2663    2712  1 ! OUTPUT PARAMETERS:
;2664    2713  1 !
;2665    2714  1 !       NONE
;2666    2715  1 !
;2667    2716  1 ! IMPLICIT OUTPUTS:
;2668    2717  1 !
;2669    2718  1 !       REC_ADDR is updated to point past the record(s) inserted
;2670    2719  1 !       REC_COUNT is incremented by the number of VBN's inserted
;2671    2720  1 !
;2672    2721  1 !
;2673    2722  1 ! ROUTINE VALUE:
;2674    2723  1 !
;2675    2724  1 !       NONE
;2676    2725  1 !
;2677    2726  1 !--
;2678    2727  1
;2679    2728  2     BEGIN
;2680    2729  2
;2681    2730  2     EXTERNAL REGISTER
;2682    2731  2         R_IRAB_STR,
;2683    2732  2         R_IFAB_STR,
;2684    2733  2         R_IDX_DFN_STR,
;2685    2734  2         R_REC_ADDR_STR,
;2686    2735  2         R_BKT_ADDR_STR;
;2687    2736  2
```

```
;  2688     2737  2     LOCAL
;  2689     2738  2         SAVE_REC_ADDR;
;  2690     2739  2
;  2691     2740  2     MACRO
;  2692     2741  2         KEY_SZ          = REC_SZ<0,16> %,
;  2693     2742  2         VBN_SZ          = REC_SZ<16,16>%;
;  2694     2743  2
;  2695     2744  2     ! Handle all empty bucket cases
;  2696     2745  2     !
;  2697     2746  2     IF .IRAB [IRB$V_EMPTY_BKT]
;  2698     2747  2     THEN
;  2699     2748  2
;  2700     2749  3         BEGIN
;  2701     2750  3
;  2702     2751  3         GLOBAL REGISTER
;  2703     2752  3             R_RAB,
;  2704     2753  3             R_IMPURE;
;  2705     2754  3
;  2706     2755  3         ! We will only swing the VBN, if the VBN of the current index record
;  2707     2756  3         ! is the same as the VBN of the leftmost primary data bucket. This may
;  2708     2757  3         ! not be the case because of a previous index update failure, system
;  2709     2758  3         ! crash, or during certain continuation bucket splits. In such cases
;  2710     2759  3         ! if we swing the VBN we could be causing crossed pointers to the
;  2711     2760  3         ! level below.
;  2712     2761  3         !
;  2713     2762  3         ! The only time this VBN swing is not done (besides when the
;  2714     2763  3         ! downpointer of the current index record is not equal to the leftmost
;  2715     2764  3         ! data VBN) is during the second pass of the two-pass multi-bucket
;  2716     2765  3         ! empty bucket split case.
;  2717     2766  3         !
;  2718     2767  3         IF .IRAB [IRB$L_VBN_LEFT] EQL RM$V3_VBN()
;  2719     2768  3         THEN
;  2720     2769  3             RM$ADD_V3VBN(.IRAB [IRB$L_VBN_MID]);
;  2721     2770  3
;  2722     2771  3         ! If there was a multi-bucket split involving empty buckets, RMS
;  2723     2772  3         ! continues the processing here.
;  2724     2773  3         !
;  2725     2774  3         IF .IRAB[IRB$V_BIG_SPLIT]
;  2726     2775  3         THEN
;  2727     2776  4             BEGIN
;  2728     2777  4
;  2729     2778  4             ! Unless this is the second pass of the two-pass multi-bucket split
;  2730     2779  4             ! case insert the key that is in keybuffer 2. If this is a one-pass
;  2731     2780  4             ! split, also increment IRB$L_REC_COUNT, the counter of the number
;  2732     2781  4             ! of VBNs succeeding the position where the VBN corresponding to
;  2733     2782  4             ! the just inserted key is to be inserted.
;  2734     2783  4             !
;  2735     2784  5             IF (.IRAB[IRB$L_VBN_LEFT] NEQA 0)
;  2736     2785  4             THEN
;  2737     2786  5                 BEGIN
;  2738     2787  5                 RM$ADD_V3KEY (KEYBUF_ADDR(2));
;  2739     2788  5
;  2740     2789  5                 IF NOT .IRAB[IRB$V_SPL_IDX]
;  2741     2790  5                 THEN
;  2742     2791  5                     IRAB[IRB$L_REC_COUNT] = .IRAB[IRB$L_REC_COUNT] + 1
;  2743     2792  5
;  2744     2793  4                 END;
```

```
;2745   2794  4
;2746   2795  4              ! Provided this is not the first pass of the two-pass multi-bucket
;2747   2796  4              ! with empty bucket split case, RMS now updates the VBN down pointer
;2748   2797  4              ! associated with the index record representing the high key of the
;2749   2798  4              ! old (left) da a bucket before the data bucket split, and its high key
;2750   2799  4              ! became the high key of the rightmost data bucket.
;2751   2800  4              !
;2752   2801  4              IF NOT .IRAB[IRB$V_SPL_IDX]
;2753   2802  4              THEN
;2754   2803  4                  RM$ADD_V3VBN (.IRAB[IRB$L_VBN_RIGHT])
;2755   2804  4
;2756   2805  3              END;
;2757   2806  3
;2758   2807  3          RETURN
;2759   2808  3
;2760   2809  2          END;
;2761   2810  2
;2762   2811  2      ! Handle all split cases not involving empty buckets.
;2763   2812  2      !
;2764   2813  2      ! We will always add at least one key provided this is not the second pass
;2765   2814  2      ! of one of the three two-pass bucket split cases. If this is one of those
;2766   2815  2      ! three cases, the key we always add was added during the first pass when the
;2767   2816  2      ! old index bucket contents were updated, and should not be again added during
;2768   2817  2      ! this update of the new index bucket contents.
;2769   2818  2      !
;2770   2819  2      SAVE_REC_ADDR = .REC_ADDR;
;2771   2820  2
;2772   2821  2      IF .IRAB[IRB$L_VBN_LEFT] NEQU 0
;2773   2822  2      THEN
;2774   2823  2          RM$ADD_V3KEY (KEYBUF_ADDR(2));
;2775   2824  2
;2776   2825  2      ! On a multibucket split, the key which will trail the two keys added must
;2777   2826  2      ! be recompressed based on the first of the two keys added. This recompression
;2778   2827  2      ! is not done if the keys being added are being added to the end of the index
;2779   2828  2      ! bucket (no trailing key), if the multibucket split case is the two-pass case
;2780   2829  2      ! where a key is added to each of the two index buckets and this is either of
;2781   2830  2      ! the two passes, or if index compression is not enabled (of course).
;2782   2831  2      !
;2783   2832  3      IF (.IRAB[IRB$V_BIG_SPLIT]
;2784   2833  3          AND NOT .IRAB[IRB$V_SPL_IDX]
;2785   2834  3          AND .IRAB[IRB$L_VBN_LEFT] NEQU 0)
;2786   2835  2          AND
;2787   2836  3          (.IDX_DFN[IDX$V_IDX_COMPR]
;2788   2837  4          AND (.SAVE_REC_ADDR + .KEY_SZ LSSA .BKT_ADDR +
;2789   2838  3                                             .BKT_ADDR[BKT$W_FREESPACE]))
;2790   2839  2      THEN
;2791   2840  2          RM$RECOMPR_KEY (.SAVE_REC_ADDR , .SAVE_REC_ADDR + .KEY_SZ);
;2792   2841  2
;2793   2842  2      ! A key and VBN must be added for all multibucket split cases. For the one-
;2794   2843  2      ! pass multibucket split case and the two-pass multibucket split case this
;2795   2844  2      ! key is the second of two keys to be added during this current pass and the
;2796   2845  2      ! first VBN. For the latter case, both keys and the VBN get added during the
;2797   2846  2      ! first pass and none get added during the second pass. For the two-pass
;2798   2847  2      ! multibucket split case where a key is inserted in each of the old and new
;2799   2848  2      ! index buckets, this key and its corresponding VBN are inserted as the low
;2800   2849  2      ! order key in the new index bucket during the second pass having inserted
;2801   2850  2      ! the first key in the old index bucket during the first pass.
```

```
;  2802          2851  2      !
;  2803          2852  2      IF .IRAB [IRB$V_BIG_SPLIT]
;  2804          2853  2          AND
;  2805          2854  4         ((NOT .IRAB[IRB$V_SPL_IDX]
;  2806          2855  4                 AND .IRAB[IRB$L_VBN_MID] NEQU 0)
;  2807          2856  3          OR
;  2808          2857  4          (.IRAB[IRB$V_SPL_IDX]
;  2809          2858  3                 AND .IRAB[IRB$W_POS_INS] EQLU BKT$C_OVERHDSZ))
;  2810          2859  2      THEN
;  2811          2860  3          BEGIN
;  2812          2861  3
;  2813          2862  3          IF .IRAB[IRB$L_VBN_LEFT] NEQU 0
;  2814          2863  3          THEN
;  2815          2864  3              IRAB[IRB$L_REC_COUNT] = .IRAB[IRB$L_REC_COUNT] + 1;
;  2816          2865  3
;  2817          2866  3          RM$ADD_V3VBN (.IRAB[IRB$L_VBN_MID]);
;  2818          2867  3
;  2819          2868  3          IRAB[IRB$V_BIG_SPLIT] = 0;
;  2820          2869  3          RM$ADD_V3KEY (KEYBUF_ADDR(3));
;  2821          2870  3          IRAB[IRB$V_BIG_SPLIT] = 1
;  2822          2871  3
;  2823          2872  2          END;
;  2824          2873  2
;  2825          2874  2      ! A VBN update is always required for one-pass bucket split cases. This is
;  2826          2875  2      ! because the VBN pointer of at least one of the index records on the current
;  2827          2876  2      ! level has changed because of the split at the level below it. For all
;  2828          2877  2      ! two-pass bucket split cases, this VBN updating must also take place, for the
;  2829          2878  2      ! same reason. However, it always takes place during the second of the two
;  2830          2879  2      ! passes when the contents of the new index bucket are being modified since it
;  2831          2880  2      ! it is the VBN pointer of one of those index records that has changed.
;  2832          2881  2      !
;  2833          2882  2      IF NOT .IRAB[IRB$V_SPL_IDX]
;  2834          2883  2      THEN
;  2835          2884  3          BEGIN
;  2836          2885  3
;  2837          2886  4          IF (.IRAB[IRB$L_VBN_LEFT] NEQU 0)
;  2838          2887  3              OR
;  2839          2888  4              (.IRAB[IRB$L_VBN_MID] NEQU 0)
;  2840          2889  3          THEN
;  2841          2890  3              IRAB [IRB$L_REC_COUNT] = .IRAB [IRB$L_REC_COUNT] + 1;
;  2842          2891  3
;  2843          2892  3          RM$ADD_V3VBN (.IRAB [IRB$L_VBN_RIGHT])
;  2844          2893  3
;  2845          2894  2          END;
;  2846          2895  2
;  2847          2896  1      END;


                              0900   8F  BB 00000 RM$V3_IDX_REC:
                                                 PUSHR   #^M<R8,R11>                    ; 2681
                    40         44   A9           06  E1 00004          BBC     #6, 68(IRAB), 5$   ; 2746
                                              0000G 30 00009          BSBW    RM$V3_VBN          ; 2767
                                    50   0088   C9  D1 0000C          CMPL    136(IRAB), R0      ;
                                                 0A  12 00011          BNEQ    1$                 ;
```

```
                                0090   C9  DD 00013          PUSHL   144(IRAB)                                  ; 2769
                                    F828   30 00017          BSBW    RM$ADD_V3VBN                                ;
                                5E             04   C0 0001A          ADDL2   #4, SP                              ;
                        03      44     A9      02   E0 0001D 1$:       BBS     #2, 68(IRAB), 3$                    ; 2774
                                    00D1   31 00022 2$:       BRW     15$                                         ;
                                0088   C9  D5 00025 3$:       TSTL    136(IRAB)                                   ; 2784
                                       17  13 00029          BEQL    4$                                          ;
                                50      00B4   CA  3C 0002B          MOVZWL  180(IFAB), R0                       ; 2787
                                    60 B940   9F  00030          PUSHAB  @96(IRAB)[R0]                           ;
                                    F78D   30 00034          BSBW    RM$ADD_V3KEY                                ;
                                5E             04   C0 00037          ADDL2   #4, SP                              ;
                                E4      44     A9  E8 0003A          BLBS    68(IRAB), 2$                        ; 2789
                                0094   C9  D6 0003E          INCL    148(IRAB)                                   ; 2791
                                DC      44     A9  E8 00042 4$:       BLBS    68(IRAB), 2$                        ; 2801
                                    00A3   31 00046          BRW     14$                                         ; 2803
                                58             56   D0 00049 5$:       MOVL    REC_ADDR, SAVE_REC_ADDR             ; 2819
                                0088   C9  D5 0004C          TSTL    136(IRAB)                                   ; 2821
                                       0F  13 00050          BEQL    6$                                          ;
                                50      00B4   CA  3C 00052          MOVZWL  180(IFAB), R0                       ; 2823
                                    60 B940   9F  00057          PUSHAB  @96(IRAB)[R0]                           ;
                                    F766   30 0005B          BSBW    RM$ADD_V3KEY                                ;
                                5E             04   C0 0005E          ADDL2   #4, SP                              ;
                        72      44     A9      02   E1 00061 6$:       BBC     #2, 68(IRAB), 11$                   ; 2832
                                2C      44     A9  E8 00066          BLBS    68(IRAB), 7$                        ; 2833
                                0088   C9  D5 0006A          TSTL    136(IRAB)                                   ; 2834
                                       26  13 0006E          BEQL    7$                                          ;
                        21      1C     A7      03   E1 00070          BBC     #3, 28(IDX_DFN), 7$                 ; 2836
                                50      0C     AE  3C 00075          MOVZWL  REC_SZ, R0                          ; 2837
                                50             58   C0 00079          ADDL2   SAVE_REC_ADDR, R0                   ;
                                51      04     A5  3C 0007C          MOVZWL  4(BKT_ADDR), R1                     ; 2838
                                51             55   C0 00080          ADDL2   BKT_ADDR, R1                        ;
                                51             50   D1 00083          CMPL    R0, R1                              ; 2837
                                       0E  1E 00086          BGEQU   7$                                          ;
                                50      0C     AE  3C 00088          MOVZWL  REC_SZ, R0                          ; 2840
                        51              58     50   C1 0008C          ADDL3   R0, SAVE_REC_ADDR, R1               ;
                                50             58   D0 00090          MOVL    SAVE_REC_ADDR, R0                   ;
                                    0000G  30 00093          BSBW    RM$RECOMPR_KEY                              ;
                        3D      44     A9      02   E1 00096 7$:       BBC     #2, 68(IRAB), 11$                   ; 2852
                                0A      44     A9  E8 0009B          BLBS    68(IRAB), 8$                        ; 2854
                                    0090   C9  D5 0009F          TSTL    144(IRAB)                                   ; 2855
                                       0A  12 000A3          BNEQ    9$                                          ;
                                33      44     A9  E9 000A5          BLBC    68(IRAB), 12$                       ; 2857
                                0E      48     A9  B1 000A9 8$:       CMPW    72(IRAB), #14                       ; 2858
                                       29  12 000AD          BNEQ    11$                                         ;
                                    0088   C9  D5 000AF 9$:       TSTL    136(IRAB)                                   ; 2862
                                       04  13 000B3          BEQL    10$                                         ;
                                    0094   C9  D6 000B5          INCL    148(IRAB)                                   ; 2864
                                    0090   C9  DD 000B9 10$:      PUSHL   144(IRAB)                                   ; 2866
                                    F782   30 000BD          BSBW    RM$ADD_V3VBN                                ;
                                44     A9              04   8A 000C0          BICB2   #4, 68(IRAB)                ; 2868
                                50      00B4   CA  3C 000C4          MOVZWL  180(IFAB), R0                       ; 2869
                                6E      60 B940   3E  000C9          MOVAW   @96(IRAB)[R0], (SP)                 ;
                                    F6F3   30 000CE          BSBW    RM$ADD_V3KEY                                ;
                                5E             04   C0 000D1          ADDL2   #4, SP                              ;
                                44     A9              04   88 000D4          BISB2   #4, 68(IRAB)                ; 2870
                                1A      44     A9  E8 000D8 11$:      BLBS    68(IRAB), 15$                       ; 2882
                                    0088   C9  D5 000DC 12$:       TSTL    136(IRAB)                                   ; 2886
```

```
                                 06  12 000E0          BNEQ    13$                                          ;
                          0090   C9  D5 000E2          TSTL    144(IRAB)                                    ; 2888
                                 04  13 000E6          BEQL    14$                                          ;
                          0094   C9  D6 000E8 13$:     INCL    148(IRAB)                                    ; 2890
                          008C   C9  DD 000EC 14$:     PUSHL   140(IRAB)                                    ; 2892
                               F74F  30 000F0          BSBW    RM$ADD_V3VBN                                 ;
                     5E          04  C0 000F3          ADDL2   #4, SP                                       ;
                          0900   8F  BA 000F6 15$:     POPR    #^M<R8,R11>                                  ; 2896
                                     05 000FA          RSB                                                  ;

; Routine Size:  251 bytes,    Routine Base:  RM$RMS3 + 0891
```

```
;2849     2897  1 %SBTTL  'RM$V3KEY_SZ'
;2850     2898  1 ROUTINE RM$V3KEY_SZ (KEY_ADDR) : RL$COMMON_LINK =
;2851     2899  1
;2852     2900  1 !++
;2853     2901  1 !
;2854     2902  1 ! FUNCTIONAL DESCRIPTION:
;2855     2903  1 !
;2856     2904  1 !    This routine computes the number of chars needed to store the
;2857     2905  1 !    prologue three key in KEY_ADDR at REC_ADDR.
;2858     2906  1 !
;2859     2907  1 ! CALLING SEQUENCE:
;2860     2908  1 !
;2861     2909  1 !       RM$V3KEY_SZ()
;2862     2910  1 !
;2863     2911  1 ! INPUT PARAMETERS:
;2864     2912  1 !
;2865     2913  1 !       KEY_ADDR - Address of the key to be inserted
;2866     2914  1 !
;2867     2915  1 ! IMPLICIT INPUTS:
;2868     2916  1 !
;2869     2917  1 !       BKT_ADDR - Pointer to the current bucket
;2870     2918  1 !       IDX_DFN  - Pointer to index descriptor structure
;2871     2919  1 !       IFAB     - Pointer to internal FAB
;2872     2920  1 !       IRAB     - Pointer to Internal RAB
;2873     2921  1 !       REC_ADDR - Address to insert key
;2874     2922  1 !                                                                  ; R
;2875     2923  1 ! OUTPUT PARAMETERS:
;2876     2924  1 !       NONE
;2877     2925  1 !
;2878     2926  1 ! IMPLICIT OUTPUTS:
;2879     2927  1 !       NONE
;2880     2928  1 !
;2881     2929  1 ! ROUTINE VALUE:
;2882     2930  1 !
;2883     2931  1 !       R0 - number of bytes needed to store the key after front compression
;2884     2932  1 !            and rear-end truncation (doesn't include two bytes of overhead).
;2885     2933  1 !
;2886     2934  1 !--
;2887     2935  1
;2888     2936  2     BEGIN
;2889     2937  2
;2890     2938  2     MAP
;2891     2939  2         KEY_ADDR : REF BBLOCK;
;2892     2940  2
;2893     2941  2     EXTERNAL REGISTER
;2894     2942  2         R_BKT_ADDR_STR,
;2895     2943  2         R_IDX_DFN_STR,
;2896     2944  2         R_REC_ADDR_STR,
;2897     2945  2         R_IFAB_STR,
;2898     2946  2         R_IRAB_STR;
;2899     2947  2
;2900     2948  2     GLOBAL REGISTER
;2901     2949  2         R_IMPURE,
;2902     2950  2         R_RAP,
;2903     2951  2         R_BDB;
;2904     2952  2
;2905     2953  2     LOCAL
```

```
;  2906          2954  2          KEYBUF  : REF BBLOCK;
;  2907          2955  2
;  2908          2956  2      ! Data level.
;  2909          2957  2      !
;  2910          2958  2      IF .BKT_ADDR [BKT$B_LEVEL] EQLU 0
;  2911          2959  2      THEN
;  2912          2960  3          BEGIN
;  2913          2961  3
;  2914          2962  3          IF NOT .IDX_DFN [IDX$V_KEY_COMPR]
;  2915          2963  3          THEN
;  2916          2964  3              RETURN .IDX_DFN [IDX$B_KEYSZ];
;  2917          2965  3          END
;  2918          2966  3
;  2919          2967  3      ! Index level.
;  2920          2968  3      !
;  2921          2969  2      ELSE
;  2922          2970  3          BEGIN
;  2923          2971  3
;  2924          2972  3          IF NOT .IDX_DFN [IDX$V_IDX_COMPR]
;  2925          2973  3          THEN
;  2926          2974  3              RETURN .IDX_DFN [IDX$B_KEYSZ];
;  2927          2975  2          END;
;  2928          2976  2
;  2929          2977  2      ! Must be a compressed key. Move key into keybuffer 5, compress it, and
;  2930          2978  2      ! return its size (not including the two byte of compression overhead).
;  2931          2979  2      !
;  2932          2980  2      KEYBUF = KEYBUF_ADDR(5);
;  2933          2981  2      RM$MOVE (.KEY_ADDR[KEY_LEN] + 2, .KEY_ADDR, .KEYBUF);
;  2934          2982  2
;  2935          2983  2      RM$COMPRESS_KEY (.KEYBUF);
;  2936          2984  2
;  2937          2985  2      RETURN .KEYBUF[KEY_LEN]
;  2938          2986  2
;  2939          2987  1      END;


                                0914    8F   BB 00000 RM$V3KEY_SZ:
                                                       PUSHR   #^M<R2,R4,R8,R11>                ;  2898
                                  0C    A5   95 00004         TSTB    12(BKT_ADDR)               ;  2958
                                        07   12 00007         BNEQ    1$                         ;
                      0D        1C    A7                    06   E0 00009         BBS     #6, 28(IDX_DFN), 3$        ;  2962
                                        05   11 0000E         BRB     2$                         ;  2964
                      06        1C    A7                    03   E0 00010 1$:     BBS     #3, 28(IDX_DFN), 3$        ;  2972
                                50      20    A7             9A   00015 2$:     MOVZBL  32(IDX_DFN), R0            ;  2974
                                        25   11 00019         BRB     4$                         ;
                                50    00B4    CA             3C   0001B 3$:     MOVZWL  180(IFAB), R0              ;  2980
                                52      60 B940              DE   00020         MOVAL   @96(IRAB)[R0], KEYBUF
                                        52   DD 00025         PUSHL   KEYBUF                     ;  2981
                                  18    AE   DD 00027         PUSHL   KEY_ADDR
                                7E      1C    BE             9A   0002A         MOVZBL  @KEY_ADDR, -(SP)           ;
                                6E            02             C0   0002E         ADDL2   #2, (SP)                   ;
                                            0000G 30 00031         BSBW    RM$MOVE                    ;
                                5E            0C             C0   00034         ADDL2   #12, SP                    ;
                                50            52             D0   00037         MOVL    KEYBUF, R0                 ;  2983
```

```
                                0000G 30 0003A          BSBW    RM$COMPRESS_KEY                         ;
                          50      62  9A 0003D          MOVZBL  (KEYBUF), R0                            ; 2985
                      0914    8F  BA 00040 4$:          POPR    #^M<R2,R4,R8,R11>                       ; 2987
                                  05 00044              RSB                                             ;

; Routine Size:  69 bytes,    Routine Base:  RM$RMS3 + 098C
```

```
;  2941     2988  1 %SBTTL  'RM$VBN_SIZE'
;  2942     2989  1 GLOBAL ROUTINE RM$VBN_SIZE (VBN) : RL$PRESERVE1 =
;  2943     2990  1
;  2944     2991  1 !++
;  2945     2992  1 !
;  2946     2993  1 ! FUNCTIONAL DESCRIPTION:
;  2947     2994  1 !
;  2948     2995  1 ! Calculate number of bytes required to describe input VBN
;  2949     2996  1 !
;  2950     2997  1 ! CALLING SEQUENCE:
;  2951     2998  1 !       RM$VBN_SIZE (VBN)
;  2952     2999  1 !
;  2953     3000  1 ! INPUT PARAMETERS:
;  2954     3001  1 !       VBN - VBN whose size is to be determined
;  2955     3002  1 !
;  2956     3003  1 ! IMPLICIT INPUTS:
;  2957     3004  1 !       NO E
;  2958     3005  1 !
;  2959     3006  1 ! OUTPUT    RAMETERS:
;  2960     3007  1 !       NONE
;  2961     3008  1 !
;  2962     3009  1 ! IMPLICIT OUTPUTS:
;  2963     3010  1 !       NONE
;  2964     3011  1 !
;  2965     3012  1 ! ROUTINE VALUE:
;  2966     3013  1 !       number of bytes needed to describe VBN
;  2967     3014  1 !
;  2968     3015  1 ! SIDE EFFECTS:
;  2969     3016  1 !       NONE
;  2970     3017  1 !
;  2971     3018  1 !--
;  2972     3019  1
;  2973     3020  2     BEGIN
;  2974     3021  2
;  2975     3022  2     LOCAL
;  2976     3023  2         SIZE;
;  2977     3024  2
;  2978     3025  2     MAP
;  2979     3026  2         VBN     : VECTOR [4, BYTE];
;  2980     3027  2
;  2981     3028  2     SIZE = 3;
;  2982     3029  2
;  2983     3030  2     DO
;  2984     3031  2
;  2985     3032  2         IF .VBN[.SIZE] NEQ 0
;  2986     3033  2         THEN
;  2987     3034  3             BEGIN
;  2988     3035  3             SIZE = .SIZE + 1;
;  2989     3036  3             RETURN .SIZE
;  2990     3037  3             END
;  2991     3038  3
;  2992     3039  2         ELSE
;  2993     3040  2             SIZE = .SIZE - 1
;  2994     3041  2
;  2995     3042  2     UNTIL .SIZE EQL 1;
;  2996     3043  2
;  2997     3044  2     SIZE = .SIZE + 1
```

```
; 2998          3045  1     END;


                                50            03    D0 00000 RM$VBN_SIZE::
                                                             MOVL    #3, SIZE                        ; 3028
                                        04 AE40     95 00003 1$:     TSTB    VBN[SIZE]               ; 3032
                                             07     12 00007         BNEQ    2$                      ;
                                             50     D7 00009         DECL    SIZE                    ; 3040
                                01           50     D1 0000B         CMPL    SIZE, #1                ; 3042
                                             F2     12 0000E         BNEQ    1$                      ;
                                             50     D6 00010 2$:     INCL    SIZE                    ; 3044
                                                    05 00012         RSB                             ; 3045

; Routine Size:  19 bytes,    Routine Base:  RM$RMS3 + 09D1

; 2999          3046  1
; 3000          3047  1 END
; 3001          3048  1
; 3002          3049  0 ELUDOM



;                                   PSECT SUMMARY
;
;       Name                    Bytes                           Attributes
;
;  RM$RMS3                       2532  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  GBL,  REL,  CON,  PIC,ALIGN(2)


;                                Library Statistics
;
;                                          -------- Symbols --------      Pages       Processing
;       File                               Total    Loaded   Percent      Mapped      Time
;
;  _$255$DUA28:[RMS.OBJ]RMS.L32;1           3109        81         2       154         00:00.4


; Information:  2
; Warnings:     0
; Errors:       0


;                                    COMMAND QUALIFIERS
;
;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RM3MISPUT/OBJ=OBJ$:RM3MISPUT MSRC$:RM3MISPUT/UPDATE=(ENH$:RM3MISPUT)
```

```
; 3003           3050  0
; Size:          2532 code + 0 data bytes
; Run Time:        00:58.1
; Elapsed Time:    02:02.7
; Lines/CPU Min:    3148
; Lexemes/CPU-Min: 15870
; Memory Used:  287 pages
; Compilation Complete
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

RM3MAKIDX
LIS

RM3FNDRRV
LIS

RM3FNDRFA
LIS

RM3GET
LIS

RM3IUDR
LIS

RM3JOURNL
LIS

RM3KEYDSC
LIS

RM3MISC
LIS

RM3MISPUT
LIS

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

RM3NEXTRE
LIS

RM3OPEN
LIS

RM3POSRFA
LIS

RM3PCKUNP
LIS

RM3POSKEY
LIS

RM3POSSEQ
LIS